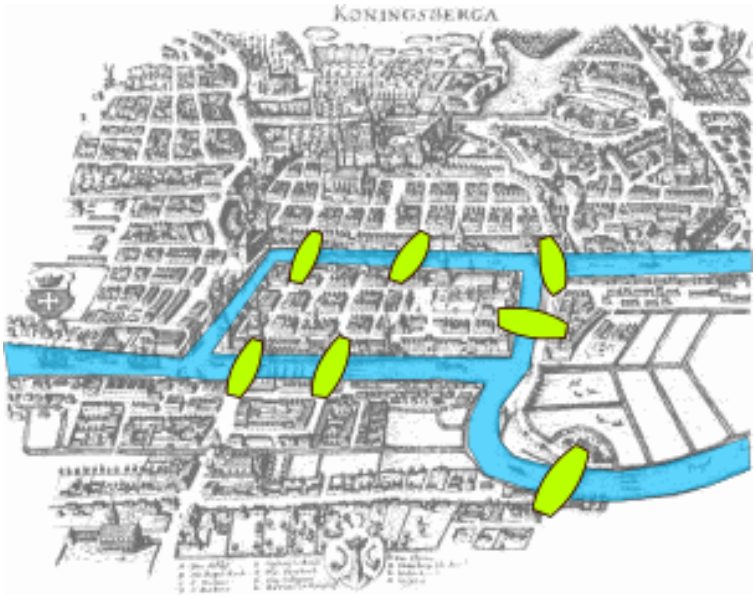
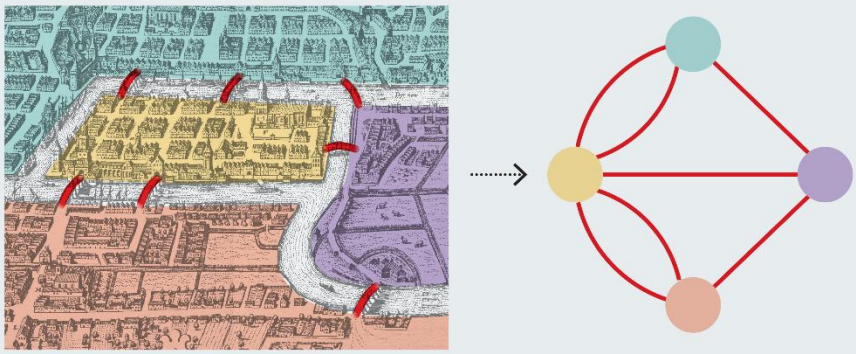
The Traveling Salesman

The “traveling salesman problem” (TSP) is a classic problem in numerical optimization that involves finding a minimum-cost path along the paths (formally, *edges*) between several points (*nodes* or *vertices*) of a network graph. This problem lies at the intersection of constrained optimization and *graph theory*, the study of networks.

A famous early example is the puzzle of the Seven Bridges of Königsberg. The city spanned a river that was crossed by seven bridges and included two islands that were fully or partially enclosed by the river. There was some debate as to whether it would be possible to take a walk through the city that involved crossing each bridge exactly once. This proved more difficult than it sounds, because of the specific layout of the bridges in relation to the island:



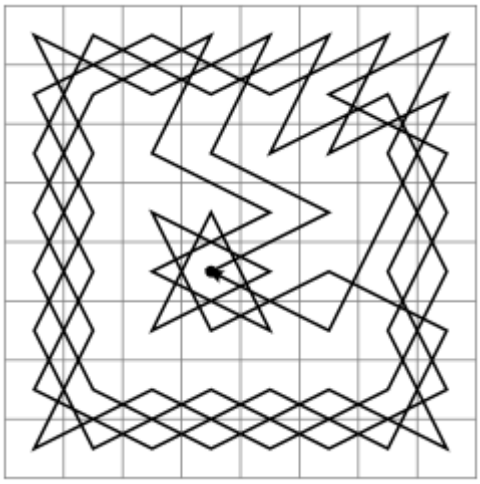
A little tinkering by hand will probably prove to you that there is no viable solution to this problem. Attempting to prove that this is the case, however, turned out to be a turning point in the history of mathematics. The legendary Swiss mathematician Leonhard Euler (1707-1783) constructed a rigorous proof that there were no possible solutions to the problem by creating an abstract representation of the city map (below) that we would recognize today as a network graph, and in so doing, invented the formal area of combinatorics known as graph theory (and also laid the foundation for the field of *topology*).



https://www.scientificamerican.com/article/how-the-seven-bridges-of-koenigsberg-spawned-new-math/

A path that covers every edge of a graph exactly once is called an *Eulerian path* in tribute to this innovation in mathematics. Similarly, a path that touches every node of a graph exactly once is called a *Hamiltonian path* (or *Hamiltonian cycle*) although it turns out that Hamiltonian paths have very different properties, and are trickier to find and evaluate.

(A famous Hamiltonian path problem that dates to antiquity, like the Königsberg bridge problem, is the problem of finding a "Knight's Tour", which is a Hamiltonian path made by a knight moving on a chessboard according to its rules of movement and visiting each square exactly once. An example is below. This may seem like trivia, but some of the early solutions to finding valid Knight's Tours were algorithms that used heuristics for guessing at a locally optimal next move – for example, choosing the space that gives the knight the fewest number of subsequent moves – that have informed our thinking about developing shortest-path algorithms for centuries.)



A "closed" Knight's Tour.

In this problem set, we are going to study variations on the TSP, which involves finding a minimum-cost path out of many possible Hamiltonian paths.

Management applications of TSP Problems

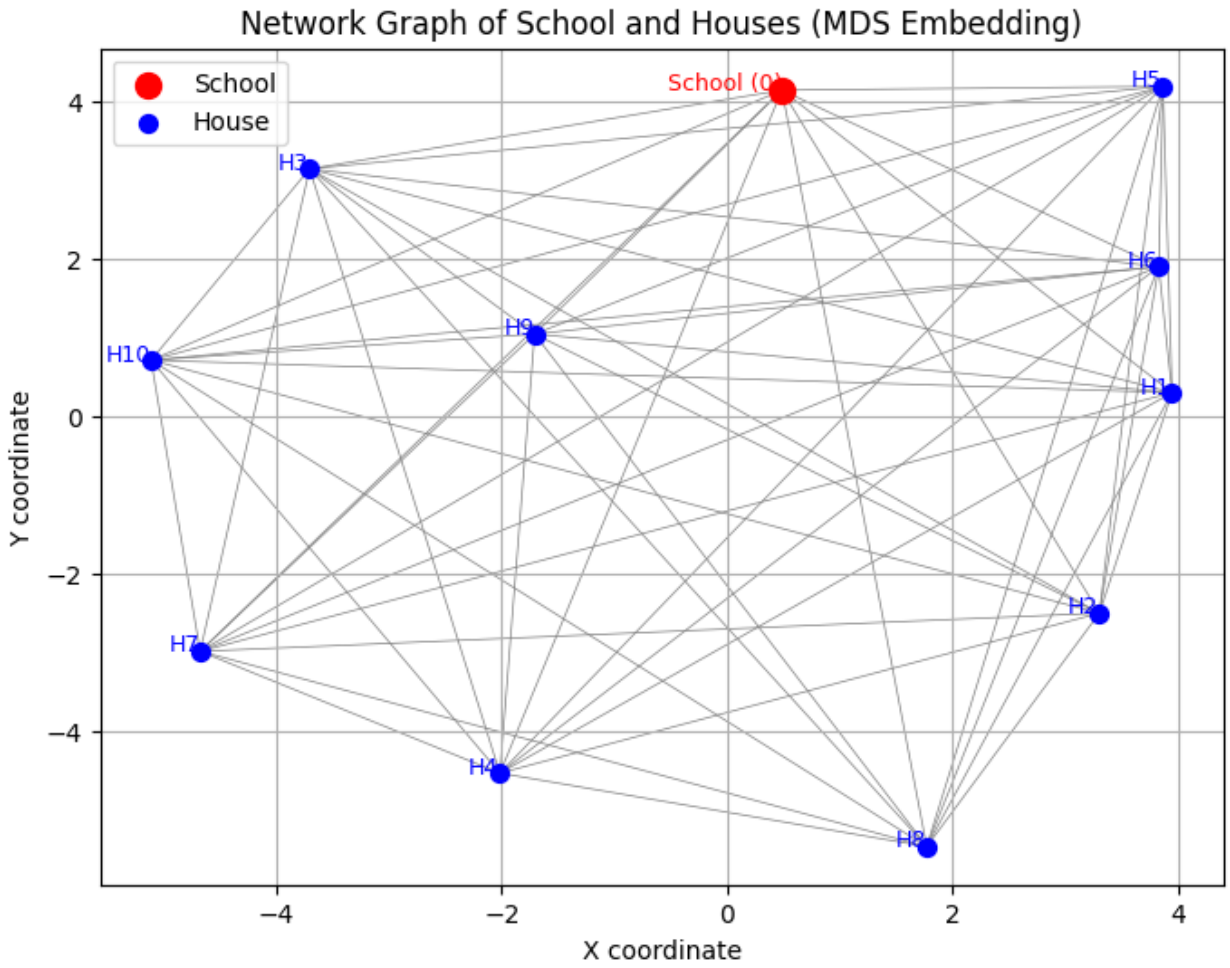
Supply chain management

Sales and field service

Municipal government

Application: School Bus Routing

For this problem set, let's assume that you are the director of transportation for a school district, and you are tasked with drawing the route map for a particular school bus that is going to leave the school, visit each of ten stops at students' houses exactly once to pick up the kids, and then arrive at school. The map of houses looks like this:



Assume that the distances shown on the graph (gray lines) are to scale.

1. Before we get into any data or formal algorithms, use pen and paper (or a digital equivalent) and draw what you think the shortest route that meets the criteria will look like, using nothing but your eyes and your intuition. Describe how you plotted your route. What kinds of "heuristics" (simple decision criteria) did you use, if any?

Next, download the "distance matrix" worksheet from Brightspace. The first tab contains the distance matrix (in miles) between each pair of houses, as well as each house and the school. Note that this is a "symmetric" matrix (the distance from A to B is the same as the distance from B to A).

2. How many possible routes (i.e., paths) exist that satisfy the criteria for the problem? Show how you found your answer.

You should find a number that is manageable for finding an exact solution to the problem (rather than an approximate solution). In this section, you will implement and compare the performance of three different algorithms that will search for an optimal solution.

3. The simplest algorithm to implement is a "brute-force" search, which evaluates all possible routes and chooses the minimum-cost path. Code up and implement a brute-force algorithm. Plot the network graph (as I have above) and highlight the path chosen by the brute-force algorithm. What is the total distance covered by your optimal path? How many iterations of the algorithm were needed to find the solution?

The next algorithm we will consider is a *dynamic programming* algorithm (Held-Karp), which approaches the problem *recursively* by breaking it into the smallest possible subproblems (i.e., finding the optimal path over smaller chunks of the overall graph). The key idea in dynamic programming is that the optimal decision at a given point is a function of the optimal decision one step ahead. This means we can solve the problem of finding the optimal strategy for maximizing a reward function (or minimizing a cost function) via the method of *backward induction*: starting at the end of the sequence of choices, finding the value of each possible choice or outcome, moving one step backwards in the sequence, and repeating the process. (We will talk more about dynamic programming in an upcoming unit.)

The Held-Karp algorithm breaks the set of all possible paths through the graph into smaller sub-problems. It asks, "if I have already visited some set of cities S, but still need to visit city k, what would be the lowest-cost way to get there?" It then stores the optimal path (known as "memoizing", or remembering, the optimal choice) for that subproblem and discards the inferior paths, which cuts down on computing time by preventing inferior

solutions from being considered repeatedly as it progresses through the problem. It then takes one step back to evaluate a slightly larger subproblem, knowing in advance what the optimal solutions are for each of the smaller subproblems contained in that larger subproblem.

4. Code up and implement the Held-Karp algorithm for our problem. Plot the network graph again, and highlight the optimal path. What is the total distance covered? How many iterations were needed to find the optimal solution? Compare this result with the result in question 3.

5. Next, we will consider an approximate solution to the problem. For a problem with ten nodes, we can find an exact solution, but as the number of nodes increases, the computing power required to find an exact solution explodes, and we quickly find ourselves in need of an approximation method. We will use the "nearest-neighbor" algorithm, which starts at the school and then always chooses the nearest unvisited house before returning to the school. Code up and implement the nearest-neighbor algorithm, plot the path it chooses on the network graph, and evaluate the performance of the algorithm in terms of the cost of the solution (i.e., the total distance covered by the path) and the number of iterations required to find it.

6. The nearest-neighbor algorithm is an example of a "greedy" algorithm. What is a "greedy" algorithm, and what are the pros and cons of these algorithms? How might a greedy algorithm be fooled into arriving at a vastly inferior solution to the globally optimal solution?

7. Another class of algorithms used in traveling salesman problems is known as "branch-and-bound" algorithms. What is a "branch-and-bound" algorithm, and what are the pros and cons?

Note that there are a variety of ways in which we could represent the "cost" of a particular route traversed by the salesman (or school bus). Up to this point, we've chosen to minimize the total distance traveled. But what if certain roads allow us to travel faster than others, and our objective is to minimize the total time that the bus spends in transit?

The second tab of the spreadsheet file on Brightspace contains the matrix of travel times (in minutes) between each pair of houses, as well as each house and the school.

8. Use an algorithm of your choosing (exact or approximate) to find the least-cost solution (in terms of total minutes traveled) to the route optimization problem. Plot your solution in a separate graph.

Fuel is a major cost center for transportation. Airlines maintain sophisticated risk management programs to hedge the cost of jet fuel. Package delivery services plot routes with an eye toward minimizing fuel expenditures; carriers like UPS and FedEx even try to minimize the number of left turns on their delivery routes in order to prevent their vehicles from idling while waiting to cross oncoming traffic. Even if a right-turn-heavy route is longer by distance, it can often be cheaper in terms of the dollar cost of fuel. We can choose to minimize the actual dollar cost of the route traveled by the bus, if we know (1) the mileage that the bus gets on each segment of the network, (2) the relationship between mileage and the speed the bus travels (they covary), and (3) the dollar cost per gallon for diesel fuel.

Assume that the relationship between school bus mileage *y* (in miles per gallon) and the speed traveled by the bus over a route segment *x* (in miles per hour) is given by:

$$y = -\frac{1}{120}(x - 60)^2 + 30$$

Note that we have assumed a maximum fuel efficiency of 30 mpg at a speed of 60 mph, which coincides with the maximum rate of speed traveled by the bus over any route segment in our data (i.e., we will not have to worry about choosing whether to drive the bus more slowly over a segment to minimize cost, as speed will always be a binding constraint on our optimization problem).

9. Using the brute-force algorithm, code up and solve the cost minimization problem to find the route that requires the minimum possible fuel expenditure. Plot the optimal route on a separate network graph. If fuel costs $4/gallon, what is the total cost of the optimal route? If you choose a nearest-neighbor approximation algorithm instead, what is the difference in total cost between the exact solution and the approximate solution?

10. Suppose that you had to consider optimizing your bus route with respect to all three versions of the problem: distance (each school bus depreciates for accounting purposes as a function of miles driven, so there is an explicit cost per mile driven in terms of a reduction in asset value); time (your bus driver gets paid hourly, and so you want to pay him or her less if at all possible); and fuel cost (as in question 9). When considering each objective separately, we arrive at three different routes. How do you think you might set up the problem to account for finding a single optimal route given all three of these overlapping objectives?