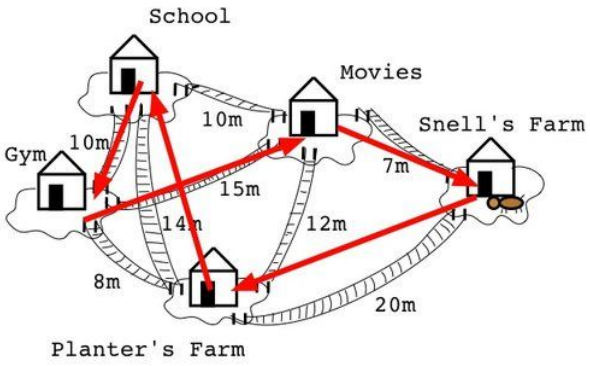
The Traveling Salesman Problem

What is the TSP?

Given a collection of cities and the cost of travel between each pair of them (cost meaning the amount of miles between them or some other way of measuring), the traveling salesman problem, or TSP for short, is to find the cheapest way of visiting all of the cities and returning to your starting point. In the standard version we study, the travel costs are symmetric in the sense that traveling from city X to city Y costs just as much as traveling from Y to X.



Background of the TSP

The simplicity of the statement of the problem is deceptive -- the Traveling Salesman Problem is one of the most intensely studied problems in computational mathematics and yet no effective solution method is known for the general case. Indeed, the resolution of the Traveling Salesman Problem would settle the P versus NP problem and fetch a $1,000,000 prize from the Clay Mathematics Institute. Although the complexity of the Traveling Salesman Problem is still unknown, for over 50 years its study has led the way to improved solution methods in many areas of mathematical optimization.

History of the TSP

Here is some history of the Traveling Salesman Problem. As aforementioned, the travel costs are symmetric in the sense that traveling from city X to city Y costs just as much as traveling from Y to X; the "way of visiting all the cities" is simply the order in which the cities are visited. To put it differently, the data consist of integer weights assigned to the edges of a finite complete graph; the objective is to find a hamiltonian cycle (that is, a cycle passing through all the vertices) of the

minimum total weight. The origins of the TSP are obscure. In the 1920's, the mathematician and economist Karl Menger publicized it among his colleagues in Vienna. In the 1930's, the problem reappeared in the mathematical circles of Princeton. In the 1940's, it was studied by statisticians (Mahalanobis, Jessen, Gosh, and Marks) in connection with an agricultural application and the mathematician Merrill Flood popularized it among his colleagues at the RAND Corporation. Eventually, the TSP gained notoriety as the prototype of a hard problem in combinatorial optimization: examining the possible paths one by one is out of the question because of their large number, and no other idea was on the horizon for a long time. By 2004, a research team was working on a TSP with a size of 24,978 cities. Below you can see some of the history and milestones related to the TSP

Year	Research Team	Size of Instance	Name
1954	G. Dantzig, R. Fulkerson, and S. Johnson	49 cities	dantzig42
1971	M. Held and R.M. Karp	64 cities	64 random points
1975	P.M. Camerini, L. Fratta, and F. Maffioli	67 cities	67 random points
1977	M. Grötschel	120 cities	gr120
1980	H. Crowder and M.W. Padberg	318 cities	lin318
1987	M. Padberg and G. Rinaldi	532 cities	att532
1987	M. Grötschel and O. Holland	666 cities	gr666
1987	M. Padberg and G. Rinaldi	2,392 cities	pr2392
1994	D. Applegate, R. Bixby, V. Chvátal, and W. Cook	7,397 cities	pla7397
1998	D. Applegate, R. Bixby, V. Chvátal, and W. Cook	13,509 cities	usa13509
2001	D. Applegate, R. Bixby, V. Chvátal, and W. Cook	15,112 cities	d15112
2004	D. Applegate, R. Bixby, V. Chvátal, W. Cook, and K. Helsgaun	24,978 cities	sw24798

P vs NP

So what makes the traveling salesman so hard. You may have heard this term used earlier, but it has a lot to do with the P vs NP problem. So what is the P vs NP problem. It is a question of interest to people working with computers and in mathematics. Basically it is a question that asks: Can every solved problem whose answer can be checked quickly by a computer also be quickly solved by a computer? P and NP are the two types of maths problems referred to: P problems are fast for computers to solve, and so are considered "easy". The P stands for polynomial time. NP problems are fast (and so "easy") for a computer to check, but are not necessarily easy to solve. The NP refers to non-deterministic time. For instance, if you have a problem, and someone says "The answer to your problem is the set of numbers 1, 2, 3, 4, 5", a computer may be able, quickly, to figure out if the answer is right or wrong, but it may take a very long time for the computer to actually come up with

"1, 2, 3, 4, 5" on its own. Mathematicians have worked out that every NP-Complete problem is reducible to every other NP-Complete problem. This means that you can rearrange the equations for TSP-DECIDE and it would look exactly the same as the factoring of prime numbers. Because each NP-Complete problem can be "reduced" to another NP-Complete problem, solving one solves them all. NP is a range of problems that could be solved in Polynomial time if only we had a special kind of Turing machine. The Non-deterministic Turing Machine works like so: every time we have to make a choice between a number of options this non-deterministic machine splits itself into as many copies as there are options. In TSP we could evaluate one city at a time but each time we come to a city we get presented with a range of choices. Our hypothetical machine would replicate itself and evaluate every one of those choices at the same time. Then for each of those choices there is another set of choices and each copy would replicate itself into even more copies. Unfortunately this machine is yet to be invented. A problem is said to be NP-Complete if it it meets these criteria: Using this Non-deterministic Turing Machine we can solve this problem in Polynomial time Using a regular Turing Machine we can verify the answer in Polynomial time. The TSP is considered to be a NP problem, thus why it hasn't been solved yet.

Brute Force Algorithm

What the Brute force algorithm attempts to do is find every single possible solution for a given graph, and then from there on it's easy to determine whether or not the solution is optimal. This is guaranteed to find the optimal solution, and in theory this sounds good… but things start to get a little tricky when the number of cities (or nodes) in the problem are increased.

Let's start with 3 nodes. There is one possible tour… easy.

With 4 nodes, there are only 3 possible tours. Not bad.

Here's where it starts to get crazy. With 5 nodes there are 12 possible tours. How can you figure this out? The way shown below was to reason that a 5th node could be added between each pair of nodes for every possible 4-node tour. Here, the actual distances don't matter because we're just trying to count the number of tours.

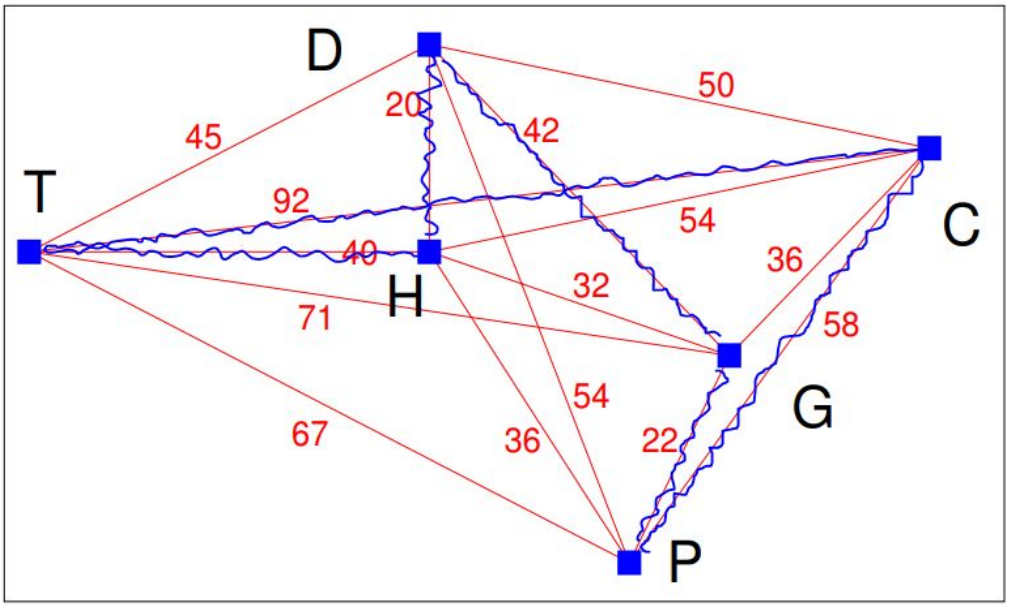
For 6 nodes we could repeat the process: add a node to every possible edge of the 12 solutions 5-node solutions. Since each of 12 solutions above has 5 edges, it means we have 12*5 = 60 tours! Notice how fast this grows: between a graph with 3 nodes and a graph of 6 nodes, our total number

of possible tours grew from 1 to 60. This problem grows very quickly!
With just 10 nodes, this grows to about 181,440 possible tours.
With just 26 nodes, this grows to a 25-digit number: 7,755,605,021,665,492,992,000,000. (By comparison, the width of the entire observable universe(!!!!) in miles is roughly a 25-digit number.)
With 100 nodes you're up to a roughly a 155-digit number.
The math: With n nodes there are (n-1)!/2 possible tours. You can confirm the formula with the examples above: 5 nodes = (5-1)! / 2 = 12.

CHALLENGE PROBLEM: Prove that for n nodes the there are (n-1)!/2 possible tours using induction

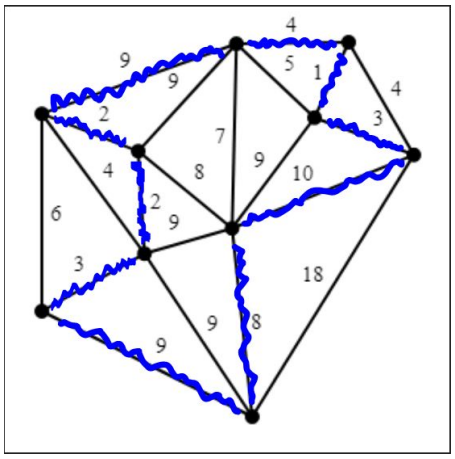
Greedy Algorithm

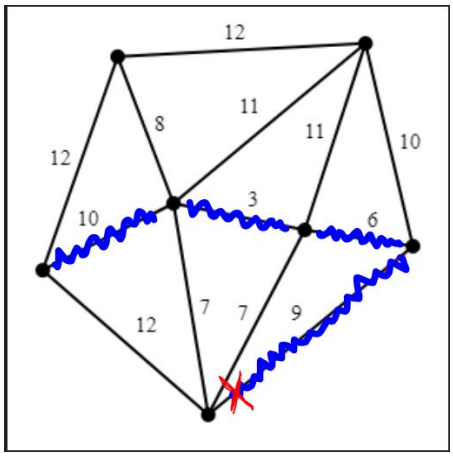
The greedy algorithm earns its name from the way that it operates. It "greedily" traverses through the edges according to the cheapest nearest node without any consideration for the most effective path as a whole. It traverses through the graph based on what is locally optimal without consideration for what is globally optimal. Now, although this algorithm runs very quickly in polynomial time and can sometimes find a close to optimal path, in most cases this algorithm isn't very useful due to a number of reasons. The first being the fact that:
It only works ALL the time with complete graphs (meaning graphs that are connected everywhere). It may not work for a graph that is not complete. Consider the graph below.



Solved using the greedy algorithm the path is H → D → G → P → C → T which has a cost of **274.** The optimum path has a cost of **229.** So, as you can see the greedy algorithm is not too far off in terms of finding a cheap path.

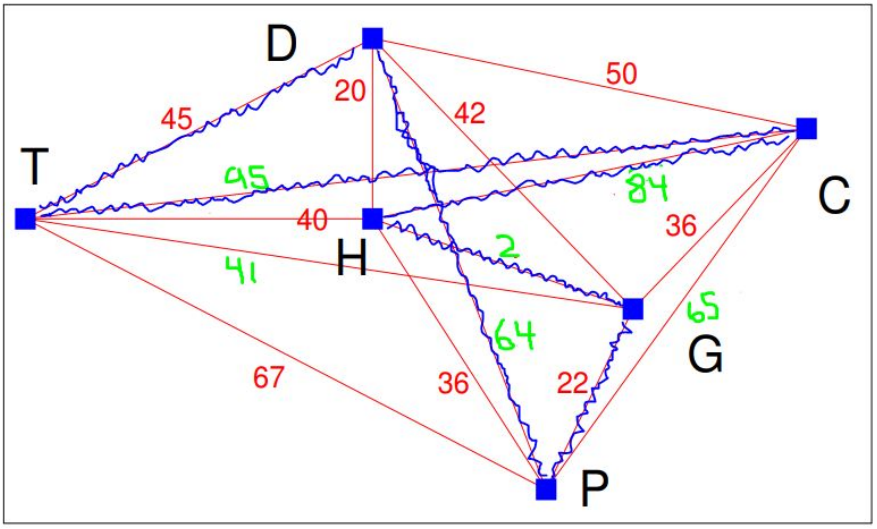
The greedy algorithm worked on the graph above because it was a complete graph. Consider these two graphs below.





Both of these graphs are not complete graphs. The one on the left works using the greedy algorithm but the one on the right encounters an issue. Once it reaches the vertex on the bottom it has no way of connecting the vertices on the top and thus can't reach every vertex to create a complete cycle. The fact that it has no guarantee of working on non complete graphs is one of the issues that makes the greedy algorithm ineffective.

Another problem with the greedy algorithm is the fact that in the same way it has the potential to find the close to optimum path, it also has the potential to find the close to the worse path for graphs with certain structures. Let's revisit our first graph but change it around a little.



As you can see we redistributed the values on some of the edges which had no effect on the optimum path which is still **229** and the overall weight of all the edges**.** However, it still had quite the effect on the greedy algorithm which drastically changed to a weight of **312** which is terrible compared to the optimum path. Despite the fact that it operates fast, complications like these prevent the greedy algorithm from being an effective tool.

Christofides Algorithm

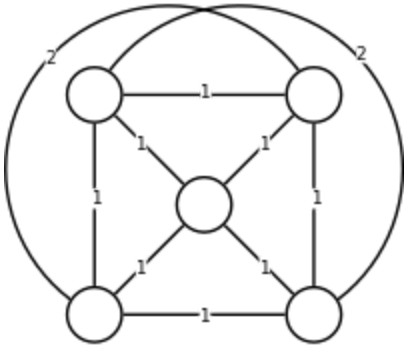
Christofides algorithm is an approximating algorithm. Meaning, it operates in polynomial time, and finds a high quality solution. The only downside to this algorithm is that for it to work, the graph must satisfy the triangle inequality meaning sum of two edges must be greater than or equal to the weight of the third edge that is connected to the two. It is proven that the cost of the solution produced by the Christofide algorithm is within 3/2 of the optimum.

Proof:

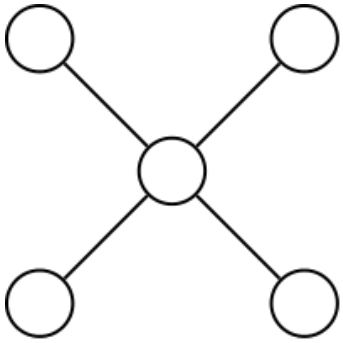
To prove this, let C be the optimal traveling salesman tour. Removing an edge from C produces a spanning tree, which must have weight at least that of the minimum spanning tree, implying that $w(T) \leq w(C)$. Next, number the vertices of O in cyclic order around C, and partition C into two sets of paths: the ones in which the first path vertex in cyclic order has an odd number and the ones in which the first path vertex has an even number. Each set of paths corresponds to a perfect matching of O that matches the two endpoints of each path, and the weight of this matching is at most equal to the weight of the paths, since these two sets of paths partition the edges of C, one of the two sets has at most half of the weight of C, and its corresponding matching has weight that is also at most half the weight of C. The minimum-weight perfect matching can have no larger weight, so $w(M) \leq w(C)/2$. Adding the weights of T and M gives the weight of the Euler tour, at most $3w(C)/2$. Shortcutting does not increase the weight, so the weight of the output is also at most $3w(C)/2$.

Here are the steps to using Christofides algorithm:

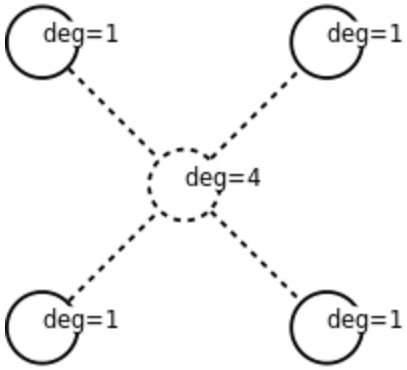
1. Given: complete graph whose edge weights obey the triangle inequality



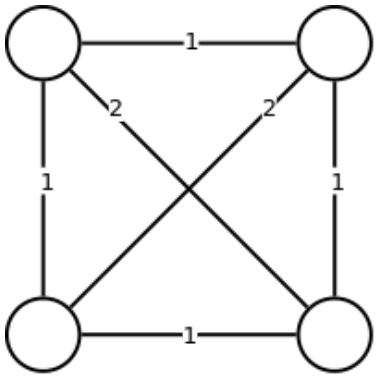
2. Calculate minimum spanning tree T



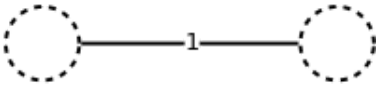
3. Calculate the set of vertices O with odd degree in T

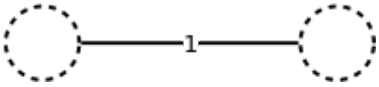


4. Form the subgraph of G using only the vertices of O

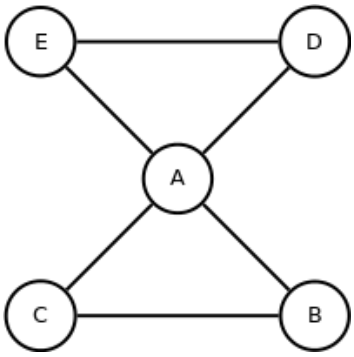


5. Construct a minimum-weight perfect matching M in this subgraph

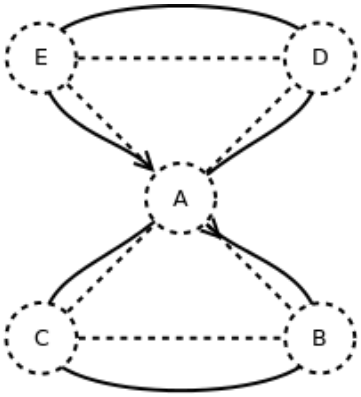




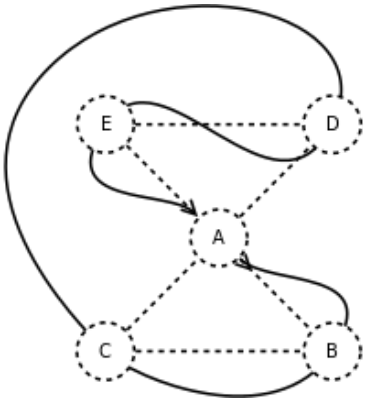
6. Unite matching and spanning tree T ∪ M to form an Eulerian multigraph



7. Calculate Euler tour



8. Remove repeated vertices, giving the algorithm's output

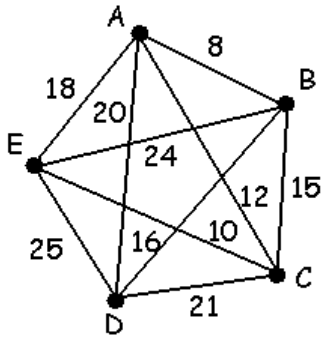


Problems

1. Prove that for n nodes the there are (n-1)!/2 possible tours using induction

2. Determine whether these problems are P or NP
    a. Greatest common divisor for any number
    b. Traveling salesman
    c. Find the minimal number of edges to add to a graph to make it Hamiltonian
    d. Given a circuit, the inputs to the circuit, and one gate in the circuit, calculate the output of that gate
    e. Find a simple path of maximum length in a given graph
    f. Maximize a linear function subject to linear inequality constraints

3. Find the optimum path starting from A first using the Brute Force algorithm, then then find an approximate using the Greedy Algorithm and Christofides Algorithm



4. The table below shows the times in minutes that it will take to travel between six places.
    a. Find the total travelling times for:
        i. The route LNOL
        ii. The route LONL
    b. Find the an approximate of the best traveling time using the greedy algorithm

	La Pedrera (L)	**Nou Camp (N)**	**Olympic Village (O)**	**Park Guell (P)**	**Ramblas (R)**	**Sagrada Familia (S)**
La Pedrera (L)	-	35	30	30	37	35
Nou Camp (N)	25	-	20	21	25	40
Olympic Village (O)	15	40	-	25	30	29
Park Guell (P)	30	35	25	-	35	20
Ramblas (R)	20	30	17	25	-	25
Sagrada Familia (S)	25	35	29	20	30	-