Data Structures & Algorithms 2

Graphs & NP problems

Lecturer: Dr. Hadi Tabatabaee
Materials: Dr. Phil Maguire & Dr. Hadi Tabatabaee
Maynooth University
Online at http://moodle.maynoothuniversity.ie

Overview

Aims

- Introduce different classes of problems.
- Introduce Traveler salesman problem.

Learning outcomes: You should be able to…

- Learn about P, NP, and NP complete problems
- Learn different approaches to solve the traveller salesman algorithm.

P-problems (easy to solve)

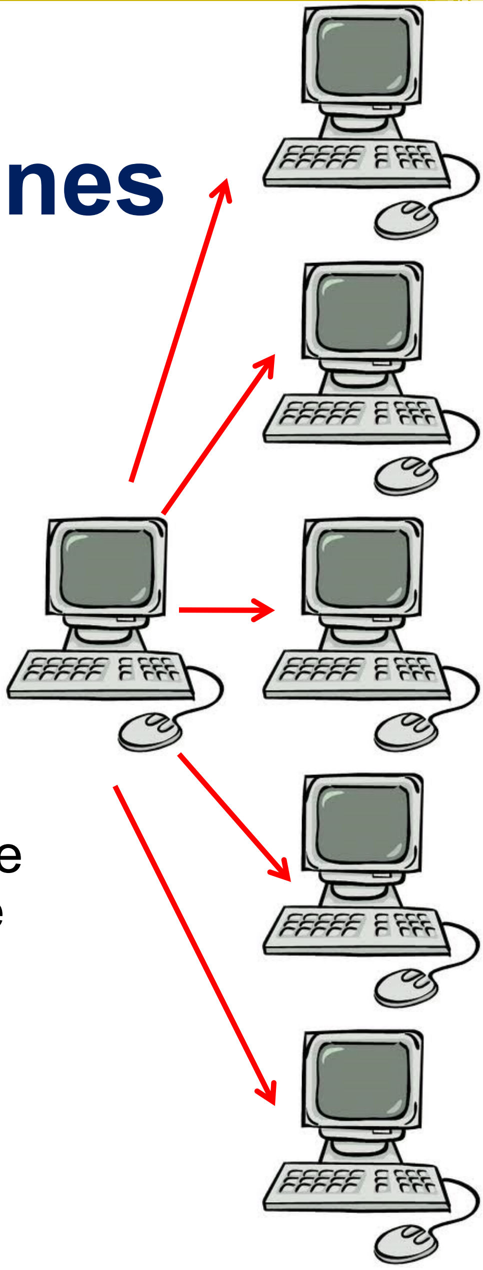
- Problems are assigned to classes depending on the complexity of the algorithm required to solve them.
- A problem is assigned to the P (polynomial time) class if at least one algorithm exists to solve that problem with a complexity of $O(n^x)$, where x is some number.
  - $n^2 + 3n$
  - $n^5 + n^3 + \log n$
  - $n^{10000} + 16$
- These are considered 'easy' problems to solve.

EXP-problems (difficult)

- A problem is assigned to the EXP (exponential time) class if the fastest algorithm to solve that problem has a complexity of $O(2^{p(n)})$, where p(n) is a polynomial expression of n (e.g., 2^{6n})
- An EXP problem is like trying to guess a password where there are no short-cuts except to try them all.
- Example: finding out if a Turing machine halts in at most *n* steps: we must try every possible set of *n* steps.
- EXP means brute force is the only way to go – we can use an imaginary non-deterministic machine to do this.

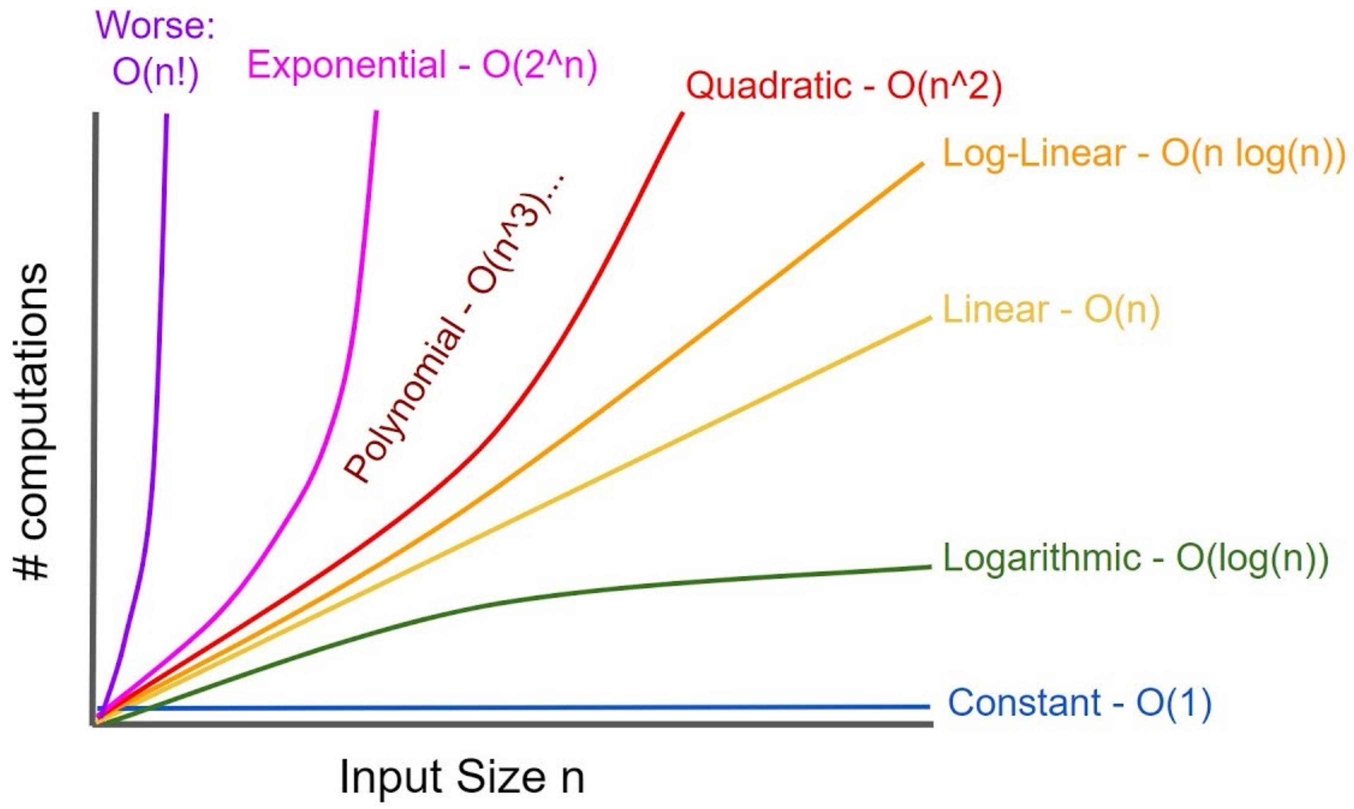
Non-deterministic machines

- A non-deterministic Turing machine is an imaginary 'magical' machine that can evaluate multiple possibilities at the same time.
- The machine branches into multiple different parallel worlds, and the solution is returned as soon as it is found by one of the branches.
- A non-deterministic Turing Machine would be able to crack a password by evaluating every possible permutation simultaneously in different worlds.
- It tries every possibility at the same time.



How bad is exponential complexity

Fibonacci example – the recursive fib cannot even compute fib(50)



NP-problems

NP is not the same as non-polynomial complexity/running time. NP does not stand for not polynomial.

- NP = Non-Deterministic polynomial time.
- NP means verifiable in polynomial time.

Verifiable?

- If we are somehow given a ‘certificate’ of a solution we can verify the legitimacy in polynomial time.

A problem is assigned to the complexity class NP if a non-deterministic polynomial machine can compute a solution, and the solution can be recognized/verified in P time by an ordinary Turing machine.

P is a subset of NP

- Since it takes polynomial time to run the program, just run the program and get a solution.

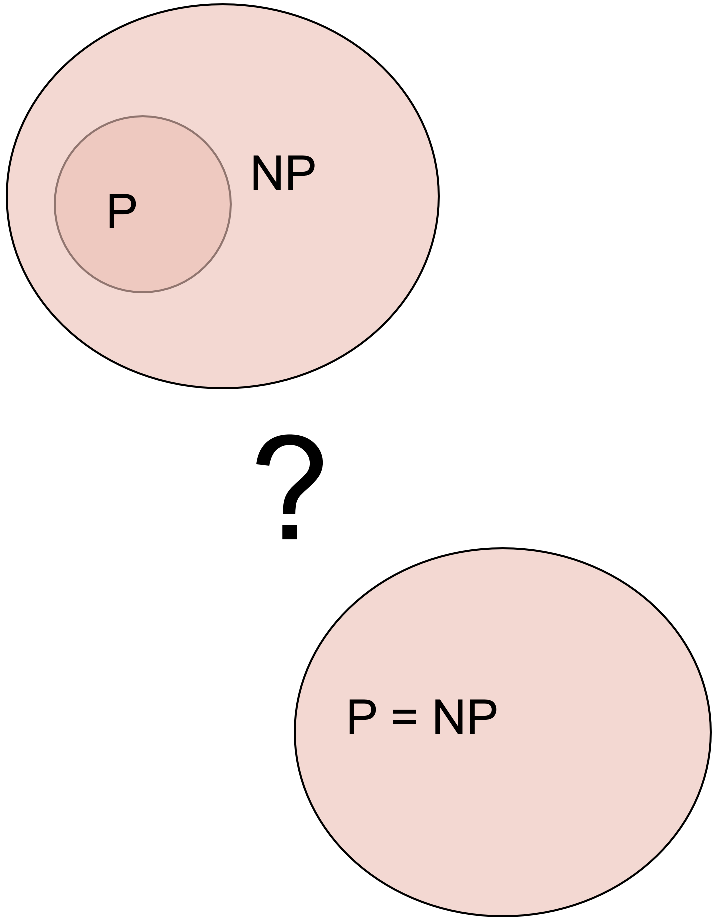
But is NP a subset of P?

- The $1,000,000 question: Are there really any problems that can be quickly recognized in P time but which cannot be solved in P time? (that is, the solution can only be computed by brute force using a non-deterministic polynomial machine)

- In other words, are there some problems in NP (answers can be easily recognized) that are not in P? (cannot be easily solved)

NP-problems

- The question P ?= NP is one of the most important questions left to be answered in mathematics.
- Are there really some problems whose solutions are easier to verify than they are to compute?
- Currently, there is no evidence either way except that some NP problems seem really difficult yet have easily verifiable solutions.



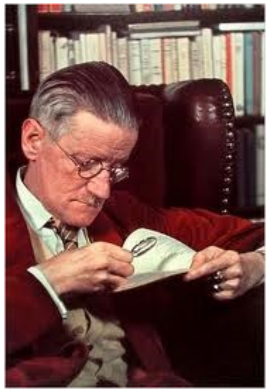
If P = NP…

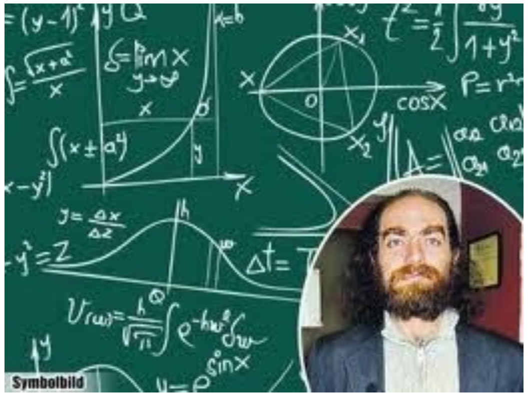
- If you could find a P solution for any NP-complete problem, it would change the fields of mathematics and computer science forever.
- Every password and code in existence could be cracked in a feasible amount of time – Internet security would be impossible.
- Every mathematical theorem with a finite proof could be proved in a feasible amount of time – all questions with answers could be answered.
- Some mathematicians believe that P != NP must be a fundamental property of the universe - if P was equal to NP, it would undermine the intuitive concepts of space and time.

P versus NP

- Examples of things that seem difficult to do yet easy to appreciate:
  - Writing a great song
  - Writing a great book
  - Proving a mathematical theorem
  - Cracking a password

- If these things were easy to do, then nothing would be valuable
  - A great book could hold no value because it would be just as easy to write it as to read it
  - No goal would be 'far away'
  - Life would be meaningless!





NP-problems

- There are many mathematical problems that have easy-to-verify solutions (i.e., are in NP) but *seem* to be difficult to solve (i.e., are not in P)

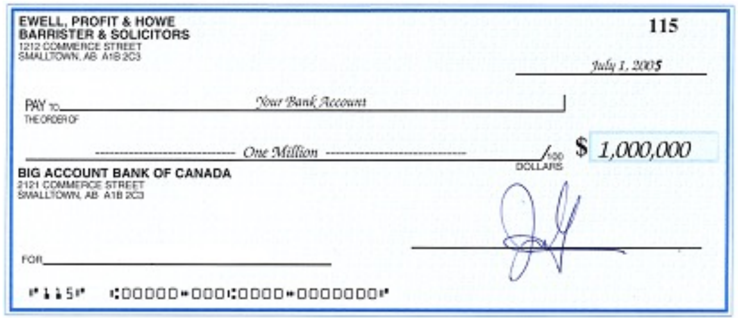
- For example, find the factors of 7,279,690,869,631
  - It is thought that solving the factorization problem is O($k^{\log n}$).
  - But maybe there is a quick way to do it?
  - A solution is 472,517 x 2,944,243
  - The solution is easy to verify.

- Can we be sure that the solution cannot be found quickly? Can we prove it?

Clay Institute Millennium Prize

- The Clay Mathematics Institute is offering a €1 million reward for answering the P =? NP question one way or the other as one of the 8 Millennium Prize problems.

- Of course, if P = NP, you could use your algorithm to find proofs for the other 7 Millennium prize problems, therefore collecting €7 million instead of €1 million!



EWELL, PROFIT & HOWE
BARRISTER & SOLICITORS
1212 COMMERCE STREET
SMALLTOWN, AB A1B 2C3

115

July 1, 2005

PAY TO THE ORDER OF Your Bank Account

One Million /100 DOLLARS $ 1,000,000

BIG ACCOUNT BANK OF CANADA
2121 COMMERCE STREET
SMALLTOWN, AB A1B 2C3

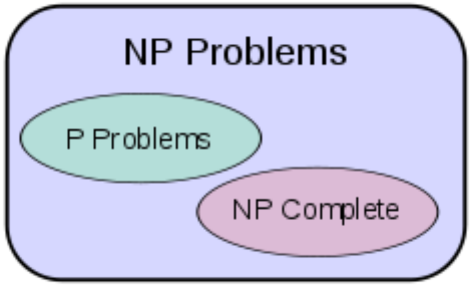
FOR

[illegible]

⑈115⑈ ⑆00000⑉000⑆0000⑉0000000⑈

NP-complete problems

- NP-complete problems are the hardest possible NP problems.
- If any NP-complete problem can ever be solved by a polynomial algorithm, then P must equal NP because it would mean that even the hardest NP problems would be easy to solve
- All NP-complete problems are really the same problem – they can be reduced to each other in polynomial time.
- Nobody has ever found a polynomial-time algorithm to solve any of them.



NP-complete example (subset-sum)

- The subset-sum problem has been proven to be NP-complete.
- Does any subset of these numbers sum to 53?
  - (15, 8, 14, 26, 32, 16, 9, 22)
- Solutions:
  - 15 + 22 + 16
  - 22 + 14 + 9 + 8
  - These solutions are hard to find yet easy to verify!

Clique problem

The problem of finding cliques (subsets of vertices, all adjacent to each other, also called complete subgraphs) in a graph.

Fixed-size clique

- An instance of a clique problem gives you 2 things as input
  - Graph
  - Some positive integer k
- The question being asked = do we have a clique of size k in this graph.

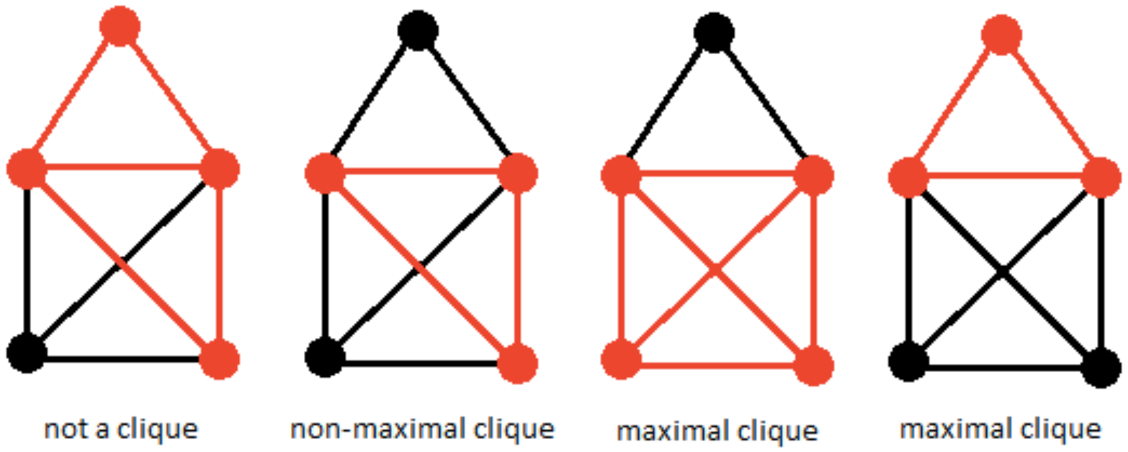
Clique problem

Analysis of fixed-size clique

- One can test whether a graph G contains a k-vertex clique and find any such clique that it contains, using a brute force algorithm.
- This algorithm examines each subgraph with k vertices and checks whether it forms a clique.
- It takes time $O(n^k k^2)$. This is because there are $O(n^k)$ subgraphs to check, each of which has $O(k^2)$ edges whose presence in G needs to be checked. Thus, the problem may be solved in polynomial time whenever k is a fixed constant.

Clique problem

A maximum clique (a clique with the largest possible number of vertices), finding a maximum weight clique in a weighted graph, listing all maximal cliques (cliques that cannot be enlarged), and solving the decision problems of testing whether a graph contains a clique larger than a given size.

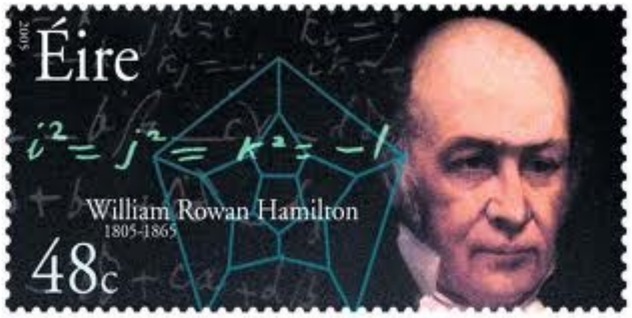


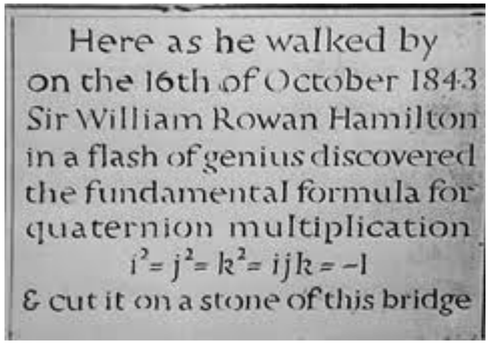
Decision problems versus optimization problems

- Finding the maximum-sized clique is an optimization problem.
- But we can reduce it to a series of decision problems
  - Can we find a clique of size 3 (why start at 3??)
  - Can we find a clique of size 4
  - Etc.

William Rowan Hamilton (1805 – 1867)

- Hamilton was an Irish physicist, astronomer, and mathematician.
- He made contributions to classical mechanics, optics, and algebra.
- In 1843 he suddenly came up with the idea of quaternions suddenly while out for a stroll along the Royal canal.
- He carved his idea into the stone of Broom Bridge in Cabra.




Here as he walked by
on the 16th of October 1843
Sir William Rowan Hamilton
in a flash of genius discovered
the fundamental formula for
quaternion multiplication
$i^2 = j^2 = k^2 = ijk = -1$
& cut it on a stone of this bridge

Traveling Salesman Problem

- The Traveling Salesman Problem (TSP) is another famous NP-complete problem that Hamilton defined.
- You're given a list of cities on a map, and you want to visit each one covering the least distance possible.
- Unfortunately, the list of possible routes is very large – it's the factorial of the number of towns.
  - 6! = 6 x 5 x 4 x 3 x 2 x 1 = 720
- The problem becomes impractical to solve for even 40 cities.

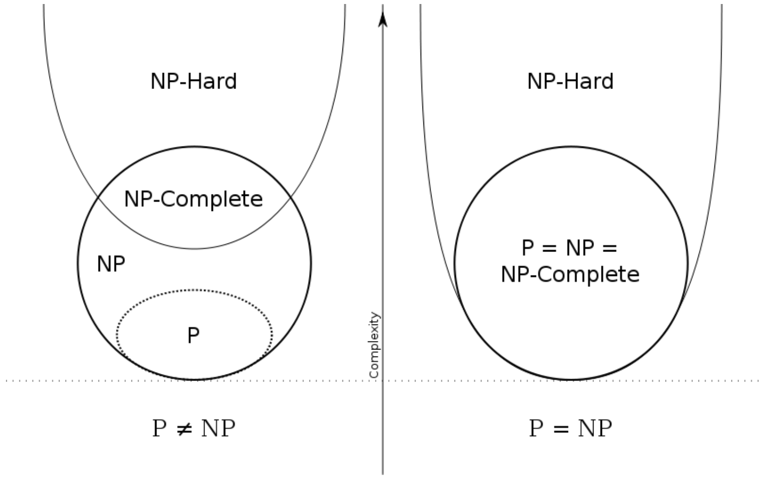
Traveling Salesman Problem

The TSP problem is the minimum spanning tree for a weighted graph problem where every vertex is linked to every other vertex, and the weights are determined by the positions of the vertices on a 2-dimensional map. In the solution, each vertex must have exactly two edges leading from it.

- You need to find the shortest path that links all of the vertices, but there are n! number of edges to consider.
- If you can find an algorithm for solving this problem whose complexity is bounded by a polynomial (e.g., n^2, n^3, n^4, etc.), then you can arrange to pick up your cheque for $1,000,000.
- Otherwise (unless you've got a few million years to spare), you'll need to design a fast algorithm that finds a good (though not optimal) solution.

TSP decision is NP-complete

- Say somebody makes a claim – *“there is a TSP path for this problem that is under 1600km”.*
- It seems difficult to verify if this is correct or not.
- However, if they show you the path, it is easy to verify if they are telling the truth.
- This is why the TSP decision problem is NP-Complete.
- The TSP optimization problem is NP-Hard, which means it is at least as hard to solve as the hardest NP problems. Still, it may not be in NP (solution is as hard as possible to compute but not necessarily easy to recognize).
- If any NP-Hard problem can be solved in P time, then so can all NP, and thus P=NP.
- For $1,000,000, can you find a quick solution for finding paths?



www.ireland-information.com
1597.517602019882KM

TSP records

- 1954 – 49 city problem solved
- 1977 – 120 city problem solved
- 1987 – 2,392 city problem solved
- 1992 – 3,038 city problem solved
- 1998 – 13,509 city problem solved
- 2004 – 24,978 city problem solved
- 2006 – 85,900 city problem solved (current record holder)

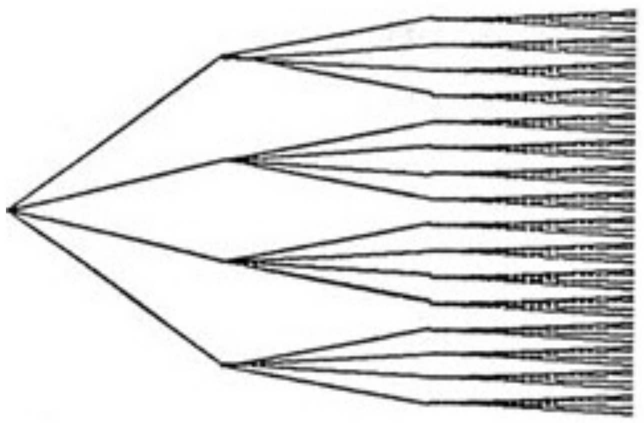
Worst Strategy

- Just pick random towns that have yet to be picked.
- You end up with a random sequence of towns.
- You might have found a good path, but it will most likely be awful.
- You could loop the program and store the best solution found so far.

Brute Force Search

- You could simply run a naïve brute-force search.
- There are 80! possible paths to be evaluated
  - That’s 7.15 x 10^{115} possibilities
  - The universe is only 4.32 x 10^{17} seconds old.
- You wouldn’t use a breadth-first search because that would only find a solution after considering every single possibility.
- The depth-first search would search all the way down, taking one particular path at each point until it finds one solution, and then it would backtrack and start finding other solutions.
- Although depth-first will quickly come up with a solution, the universe would expire before the best solution is found.

Branching Factor



- The above tree has a branching factor of 4 (4 options at each level).
- Chess has a branching factor of around 35.
- The project has a branching factor of 80!
- We need some heuristics to speed up the search.

Greedy algorithms

- We could use a greedy algorithm like the nearest neighbor.
- Keep choosing to go to the nearest town first.
- Just use your matrix and mark out towns you've visited, so they are not revisited (you can do this by setting all the edges leading to that town to a huge number, like a million kilometers, so they never get selected)
- You'll probably find that you finish up quite a distance from Maynooth at the second last step.

A clever random idea

- Let the computer do the work rather than your brain.
- Combine the greedy search with a random element.
- For example, add a random number from 0 – 20km to all the slots in your matrix.
- Now a slightly different answer will be generated each time.
- Just keep track of the best solution found so far and let your algorithm run overnight!

A better greedy algorithm

- Another alternative is to rank all of the distances between all towns.
- Keep adding in the shortest distance between any two unvisited towns on the map.
- The one condition is that you want to doubling back on yourself.
  - Don’t include links to towns you have included already.
  - For example, you don’t want to end up back in Maynooth before completing a full tour.

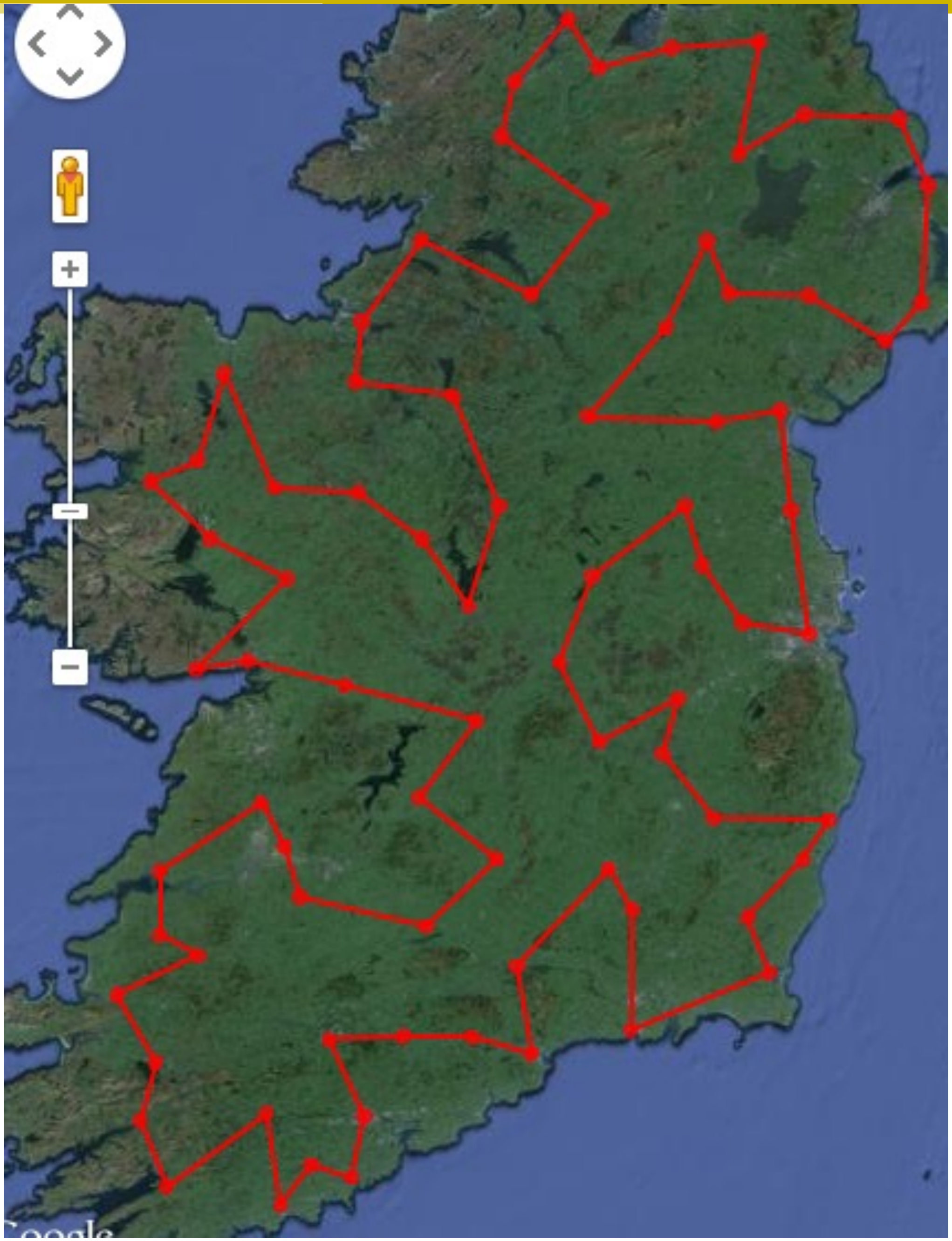
Held-Karp

- Proposed in 1962 by Bellman and independently by Held and Karp.
- It is based on dynamic programming.
- Dynamic programming is a method for solving a complex problem by breaking it down into a collection of simpler subproblems.
- Dynamic programming avoids resolving the same sub-problems over and over again by storing the solution in memory in lookup tables.
- It is an example of a space-time tradeoff.

Held-Karp

- Every sub-path of a path of minimum distance is itself of minimum distance.
- Compute the solutions to all subproblems, starting with the smallest.
- Whenever computation of a solution requires solutions for smaller problems, look up these solutions that are already computed.
- The time complexity is $O(2^n n^2)$, and the space is $(2^n n)$.

Held-Karp



Held-Karp

- You encode sub-solutions in terms of where you are now, and which towns have left to be visited

$$
\begin{aligned}
g(1,\{2,3,4\}) = \min \{ & c_{12} + g(2,\{3,4\}), c_{13} + \\
& g(3,\{2,4\}), c_{14} + g(4,\{2,3\}) \} \\
& = \min \{2 + 20, 9 + 12, 10 + 20\} \\
& = \min \{22, 21, 30\} \\
& = 21
\end{aligned}
$$

Linear Programming

- TSP can also be formulated as an integer linear program
- Label the cities with numbers 0 to n and define

$$x_{ij} = \begin{cases} 1 & \text{the path goes from city } i \text{ to city } j \\ 0 & \text{otherwise} \end{cases}$$

- Now we just need to solve the matrix, with the constraints that
  - 1) each city be arrived at from exactly one other city.
  - 2) from each city there is a departure to exactly one other city.
  - 3) the tour isn’t broken up into separate ‘islands’.

Linear Programming

- You need to import software for solving linear equations.
- CPLEX made by IBM has a Java interface and is very fast.
- See this video for using Java and CPLEX https://www.youtube.com/watch?v=sf59_7r8QSY

$$\min \sum_{i=0}^{n} \sum_{j\neq i, j=0}^{n} c_{ij} x_{ij}$$

$$\begin{array}{ll}
0 \leq x_{ij} \leq 1 & i, j = 0, \cdots, n \\
u_i \in \mathbf{Z} & i = 0, \cdots, n \\
\displaystyle\sum_{i=0, i\neq j}^{n} x_{ij} = 1 & j = 0, \cdots, n \\
\displaystyle\sum_{j=0, j\neq i}^{n} x_{ij} = 1 & i = 0, \cdots, n \\
u_i - u_j + n x_{ij} \leq n - 1 & 1 \leq i \neq j \leq n
\end{array}$$

DFS with Branch-and-Bound

- Branch-and-bound is a heuristic we can use to speed up the depth-first search by not bothering to consider paths that couldn’t possibly end up being better than the best solution found so far.
- If a partial tour is already longer than the best solution found so far that there is no need to continue searching down that path

Upper Bound

- Run your depth-first search, which quickly comes up with a solution.
- Keep track of the best solution found so far and the distance involved (e.g., 2,140 km).
- If your algorithm is examining a partial solution that already exceeds 2,140km, then there is no point in continuing to search that path.
- Backtrack and pop the next partial solution off the stack.

Better Branch-and-Bound

- We can make branch-and-bound even better by figuring out the lower bound for a partial path – the minimum size that any full solution will be based on a partial path.
- We can then dump any partial paths for which the lower bound already exceeds the upper bound.
- We don’t even have to wait for a partial path to exceed the upper bound before dumping it if we know that it will definitely exceed the upper bound by the time the path is complete.
- Obtaining the lower bound is done by examining the distance matrix and transforming it in some clever ways.

Obtaining a Lower Bound

- We use the distance matrix for the towns remaining to be visited.
- We can subtract a constant from any row or column without affecting the validity of the solution.
- The outputted weights will change, but the resulting solution will still be correct.

i\j	1	2	3	4	5	6	7
1	Inf	3	93	13	33	9	57
2	4	Inf	77	42	21	16	34
3	45	17	Inf	36	16	28	25
4	39	90	80	Inf	56	7	91
5	28	46	88	33	Inf	25	57
6	3	88	18	46	92	Inf	7
7	44	26	33	27	84	39	Inf

Obtaining a Lower Bound

- We normalize the matrix by subtracting the minimum value in each row from the row and the minimum value of each column from the column
- This results in a matrix with at least one zero in every row and column

i\j	1	2	3	4	5	6	7
1	Inf	0	83	9	30	6	50
2	0	Inf	66	37	17	12	26
3	29	1	Inf	19	0	12	5
4	32	83	66	Inf	49	0	80
5	3	21	56	7	Inf	0	28
6	0	85	8	42	89	Inf	0
7	18	0	0	0	58	13	Inf

Obtaining a Lower Bound

- Any solution must use one entry from every row and every column, so the sum of the values we just subtracted is a lower bound on the size of the eventual complete path.
- In this case we subtracted [3 4 16 7 25 3 26] from the rows and then [0 0 7 1 0 0 4] from the columns.
- Without needing to evaluate the remainder of the path, we know there is no way the path could possibly be less than 96.
- If this already exceeds the upper bound, then we can avoid considering any more possibilities involving this partial path.

Obtaining a Lower Bound

- What have we actually done here?
- We’ve found the shortest distance taken to reach each city from any other city.
- We’ve assumed that every city will be reached from the city nearest to it and figured out the cumulative distance in this case.
- In reality, any actual solution will involve a longer distance than this.
- As you build up your partial path, the rows and columns corresponding to towns you have already visited will become redundant.
- This means that the lower bound will constantly change – it will increase as the path gets longer.

Questions