Comparative Analysis of Green Vehicle Routing Problem

Submitted By:

Anushka S, 2019A7PS0208U

Dhakshina Priya I, 2019A7PS0003U

Asmi S, 2019A7PS0147U

Abstract:

The Transportation sector is a major contributor to the emission of Greenhouse Gases which can have devastating effects on the environment as well as accelerate global warming. With added global pressure and directives to enforce sustainable practices, environmental concerns form a crucial part of corporate social responsibility policies. It also aids as a competitive advantage. Therefore, there is a need for efficient models which can help reduce carbon emissions. This paper consists of a detailed literature review of the Green Vehicle Routing Problem (GVRP). The Vehicle Routing Problem (VRP) is an integer programming and combinatorial optimization problem. It is a generalization of the popular traveling salesman problem and comes under the domain of NP-Hard problems - problems whose solutions are easy to verify but hard to find an algorithmically efficient formula to. The classical VRP deals with the delivery of goods from a depot to a set of customers in various cities. Each vehicle route must start and end at the depot after visiting all the cities exactly once. The main objective of VRP is to minimize total distance or time travelled. There are many variations of VRP depending on the constraints imposed. One of the variants is GVRP which introduces the constraint of minimizing fuel consumption or carbon emissions. Another class of GVRP is Pollution Routing Problem (PRP) which also aims to find the vehicle path which would lead to the least amount of pollution. One main issue with GVRP is the uncertainty with respect to the parameters such as customer demand or travel time. As these parameters can vary based on usual daily events. The solutions to the GVRP problem must be both scalable and general enough to adapt as solutions to multi-objective variants of this problem. This paper constructs a thorough comparative analysis by reviewing the various algorithmic approaches to the problem, and their implementations. The proposed solutions are compared with existing solutions to compute the performance measure values and the advantages and disadvantages of said algorithms on varying use cases.

Problem definition:

With rising global pressures and constraints to enforce sustainability and reduce emissions in heavily polluting sectors like the transportation sector, a green solution is the need of the hour. The main objective is to find an optimum solution to the extended VRP, precisely the GVRP that aims to minimize carbon emissions while tackling the VRP. Thus, the requirement is a model that would aid in designing the transportation network which targets reducing vehicle fuel consumption while cutting down on operational costs and achieving performance indicators like customer satisfaction.

Objectives:

1. To understand the GVRP, its importance and the need to solve it.

2. To research the various proposed solutions for the GVRP in recent literature.

3. To determine the objectives, problem statement, methodology, dataset, algorithm, advantages, disadvantages, and performance measure for each reviewed GVRP solution.

4. To learn how solutions are formulated and implemented for real world problems and further, to gain an understanding of how to measure the effectiveness of one's approach.

5. To grasp how algorithms from different domains can be brought together to create more efficient techniques for solving issues with greater degrees of complexity.

Literature Table:

Reference	Objectives	Problem Statement	Methodology	Dataset	Algorithm	Advantage	Disadvantage	Performance Measure Value
[1]	To use RDP-SOC Algorithm to determine the vehicle paths where the vehicles stop by each customer’s location exactly once and whose path should begin and end at the depot in such a way that the total cost is reduced.	Finding a heuristic approach for GVRP with the constraints of minimizing total costs which includes cost of fuel consumption and driver’s cost.	The DP based solution approach is unlike the other variants of DP based algorithms since it can be used to estimate amount of fuel consumed based on the distance travelled, speed of the vehicle and the features of the vehicle while construction of the delivery plan. The RDP-SOC, classical DP, Restricted DP (RDP) and Simulation based RDP (SRDP) algorithms are compared with each other with respect to their performance on base case, two	The Pollution Routing Problem Instance Library and The Routing Problem Library	Restricted Dynamic Programming with Simulation and Online Control (RDP-SOC)	Proposed algorithm can accommodate several key performance indicators. Large sized routing problems can be solved in shorter computation time.	The proposed algorithm assumes An average speed of vehicles in separate delivery routes which causes a 0.5 % increase in the cost when heterogeneity in arcs is assumed. It also finds out completely different vehicle paths for individual vehicles when the actual speed of vehicles is taken into account as opposed to the average speed.	When compared to SRDP the proposed algorithm achieves 1% lower average cost. The computation time of the proposed algorithm is almost half of the Restricted Dynamic Programming Algorithm (RDP).

			small-sized problems and a set of larger sized problems.					
[2]	To use NSGA-II to minimize the distance travelled by the vehicles as well as minimize the amount of carbon emission while visiting all the customer's location exactly once.	To solve Green Mixed VRP with the intent to minimize the distance travelled and carbon emissions as well as considering the customer demand in each city to be a rough variable.	The NSGA-II algorithm first randomly generates 100 chromosomes in accordance with the vehicle capacity. The population size used is 100. Next in the selection step the parents are selected. Then a cross over technique is used in order to produce partial offspring. Finally, mutation is introduced using swapping method.	VRP Library [3]-[5], Set P Dataset [6] and dataset for rough variable [7]	Non-Dominated Sorting Genetic Algorithm II (NSGA-II)	The proposed algorithm achieves better results due to selection of a good range of outputs and good convergence close to the non-dominated results. The work done in the paper also is more relevant to real-life situations.	The algorithm's performance on larger problems is not defined in the study.	VIKOR method is used to get results close to ideal solution. For multi-objective mixed G-VRP model the decision maker chose the P-n23-k8 instance which is 386.759003 km distance and 1995.332031 kg carbon dioxide emissions.
[8]	To study the impact of uncertainty in RPRP by using ALNS algorithm which is adapted for hard worst case	To tackle the issue of uncertainty of input data to achieve robust optimization model to deal with PRP. And to therefore	The ALNS algorithm works on idea of bettering the initial solution by using destroy and repair operators.	The Pollution-Routing Problem Instance Library [9]	Adaptive Large Neighbourhood Search (ALNS) Algorithm	The proposed robust model helps to show the impact of uncertainty in large scale instances and	There are constraints on the capacity of the vehicles as well as the minimum speed levels that must be	Using the proposed approach, for each group of instances and different uncertainty

	robust optimization of the model.	reduce the cost for fuel and driver of vehicle.	In each iteration a removal operator which destroys and an insertion operator which repairs the solution are chosen by a roulette wheel mechanism in accordance with past performance. The new solution is approved if it meets a certain criterion. Then the score associated with the operators is updated and finally a speed optimization procedure is employed to calculate optimal speed at each arc which in turn minimizes the cost of fuel consumption and driver wages.			help decision makers choose the models with suitable uncertainty levels.	maintained the vehicles.	levels its impacts on cost of solution, total distance of routes and number of vehicles used have been examined. The results show in comparison with deterministic cost for 10-node instances there is a 14.47% increase in solution cost of 50% uncertainty level. And this only further increases to 31.45 % for 200 instance nodes. This clearly shows that the data uncertainty in PRP model leads to an

								increase in the objective functions especially for large sized problems.
[10]	To extend the Branch and Fix algorithm, an exact method, and embed it in a stochastic process to create a heuristic approach that will compute a Pareto set of solutions that solve the multi objective Hamiltonian cycle problem (the most elementary form of the GVRP) by minimizing the sum of the weights of the arcs for each objective To measure the efficiency of the proposed algorithm by	The Hamiltonian cycle problem involves finding a cycle in a given graph that touches every vertex exactly once or determining that this is not possible. In this paper [1], a graph is considered along with a set of matrices. Each matrix puts different weights on the edges of the graph. Hence, the graph along with these matrices form a multi objective Hamiltonian cycle problem. The goal is to formulate an algorithm that can find a set of Hamiltonian cycles	The heuristic uses an algorithm that takes the adjacency matrix of a graph and the associated weight matrices (for each objective) as input and returns the Pareto Set of Hamiltonian cycles as output. A Markov Decision Process is established where branches take place at every node (the root being the starting vertex) to generate options for the next edge to be added to form a cycle. At most two linear programs are	The MO-BF was tested on 500 randomly generated graphs (having defined densities and number of vertices that diversify the test set) The first 10 graphs from the FHCP challenge set	Multi Objective Branch and Fix Algorithm (MO-BF)	In comparison to the Multi Objective Genetic algorithm, the MO-BF algorithm performs very well for sparse graphs and challenge set graphs (i.e., it works well with graphs having low density and high number of vertices) The MO-BF is an anytime algorithm, i.e., it can be stopped at any time during its execution to retrieve the set	In comparison to the Multi Objective Genetic algorithm, the MO-BF algorithm performs worse for dense graphs where it is usually easier to find Hamiltonian cycles. This is probably due to the algorithm spending more time in suboptimal branches given a larger number of edges. The algorithm evolves the quality of its solution set over time. This means that longer runtimes (which may not be	Performance was measured based on mean HV (Hypervolume) over multiple runs of MO-BF (and compared to mean HV over the same amount of runs of MO-GA) for all solved instances. MO-BF dominated MO-GA for random graphs with density 0.15 having 60, 70, 80, 90, 100 vertices, for random graphs with density 0.25 having 60 vertices and for all 10 challenge

	comparing it with a multi objective genetic algorithm in graphs of different number of vertices and density	in the graph where the edges chosen effectively to minimize the weights imposed by all the objectives put together.	solved for every branching node to check if the solutions being generated are feasible and non-dominated by solutions in other subtrees. Any branches violating these conditions are pruned. A permutation-based approach is also tested where the vertices and their corresponding weights are shuffled in the input matrices before execution to allow the algorithm to begin from different nodes. In this case the algorithm is run parallelly in different threads for every permutation of node ordering.			of solutions it has generated so far Since it solves the most elementary sub-problem of the GVRP, the MO-BF can be easily generalized to form a solution for it The permutation-based MO-BF can consider more possible solutions for smaller graphs and thus give better results	practical) may be required for it to generate better solutions.	graphs considered. The literal values are presented in [10, Tab. 2].

[11]	To propose a two-phase solution approach for the GVRP that breaks down routes into a composition of paths that are combined to create feasible solutions To test the proposed solution on small to medium sized benchmark instances and verify whether it outperforms existing exact methods in terms of solution optimality and computational time. To convert this approach into a heuristic and test its efficiency on larger sized instances	The GVRP aims to route AFVs based at a single depot, to handle customers while minimizing the total distance travelled. Due to limited fuel capacity, they may need to stop at AFSs during their trip. A feasible solution to this problem is one that generates a route for each available vehicle covering the maximum number of customers while making minimal refuel stops in the shortest possible time. An algorithm must be proposed to generate various such solutions while having an inexpensive computational complexity	The solution is formulated by considering each route to be a combination of paths. The method involves two phases: All feasible and non-dominated paths formed by node pairs are generated. Some paths are combined, using a Mixed Integer Linear Programming formulation to put them into sequence. The objective is to minimize the total distance travelled. Feasibility, dominance and compatibility rules at each stage ensure that dominated paths are removed and optimal paths are	The proposed exact approach has been tested on the benchmark instances (sets S1, S2, S3 and S4) introduced in [19]. The heuristic approach has been tested on the larger set of instances AB (AB1 and AB2) introduced in [20].	Path Based Approach for the GVRP	In comparison to eight other previously proposed methods the Path Based Approach has given better quality solutions in lesser computation times for smaller sized problems Due to its bottom up approach the algorithm is able to consider a wider range of solution options compared to other exact methods	The solution quality of the Path Based Approach degrades as graph size increases, i.e., it does not scale very well	Performance was measured on the basis of total distance travelled and CPU time taken for each solution by the Path Based Approach. The same was noted for solutions generated by eight other algorithms for performance comparison. PBA outperformed each of them when tested on instances S1, S2, S3 and S4 of [19]. The literal values are presented in [11, Tab. 5], [11, Tab. 8] and [11, Tab. 9].

			designed. This solution is also reformulated as a heuristic by reducing the number of solutions generated in the first phase to handle larger sized problems.					
[12]	To reformulate the GVRP as a multi graph problem and define a three phase multi start local search algorithm to solve it To test the proposed algorithm on benchmark instances to assess its competitiveness	The GVRP is composed of customers, refuelling stations, and AFVs from a central depot. The objective is to plan routes such that each customer is visited exactly once, and the total distance travelled by the AFVs is minimal. The vehicles have restricted driving range and thus must stop at refuelling stations, which incurs a delay.	The GVRP is first reformulated as a multigraph. Then, a multi start local search heuristic is applied to it. The solution set obtained from this phase is improved upon through the local search phase. Intensification and diversification phases are then introduced to further modify the solution set. Finally, a set partitioning heuristic works on	Two sets of benchmark instances were considered: The first set was proposed in [19]. It has larger instances containing 111–500 customers and 21–28 refuelling stations. The proposed algorithm was tested on these larger instances.	Multi Start Local Search Heuristic	The algorithm is able to consistently find solutions of very high quality across instances of various types and sizes	The algorithm trades off CPU time for solution quality. It takes a longer time to execute even for instances of smaller sizes. Compared to other proposed methods/heuristics, this solution requires a far higher number of global iterations to reach an answer	The performance of the algorithm is measured on the basis of the best-known solution cost generated by it (in comparison to the best costs found by the authors of the datasets used for testing) and computation time. It outperforms the original findings for a majority of the

		Finally, each vehicle’s trip has a maximum duration constraint. A metaheuristic model must be developed to solve the GVRP as described.	improving solution quality. Various operators tailored to multigraph operations (such as Clarke and Wright savings, 2-OPT*, 3-OPT*, sequence relocate, cyclic exchange, intra-route 2-OPT, and intra-route relocate operators) are used throughout all these phases to form optimal solution paths.	The algorithm was also tested on the AB set of instances [20].				instances discussed. The literal values are presented in [12, Tab. 5], [12, Tab. 6] and [12, Tab. 7].
[13]	To propose a Memetic Algorithm with Competition (MAC) to solve the Capacitated GVRP To test the impact of each component of the algorithm on its efficiency	In the CGVRP customers are serviced by several AFVs that start and end their route at the depot and have the same loading capacity. The total demand for one vehicle cannot exceed its capacity. The vehicles cannot	MAC uses k-nearest neighbour (kNN) heuristic for population initialization. The next phase is intensification for which variable neighbourhood search (VNS) and simulated annealing (SA)	The algorithm was tested on two groups of CGVRP instances: For the group with small instances, the number of customers ranges from 15 to 24, and the	Memetic Algorithm with Competition	The MAC algorithm uses a decoding method which allows many factors (i.e., modifications to the original problem statement) to be adjusted and hence it can	Since Alternative Fuel Stations (AFSs) can be visited more than once in a graph by different vehicles, the algorithm generates a set of dummy AFS vertices for the route of each vehicle that visits	The performance of the MAC algorithm was measured in terms of CPU time consumed and distance from the optimal solution for each instance

	To determine suitable values for algorithm parameters To perform extensive comparative analysis of the proposed algorithm to estimate its effectiveness in relation to existing methods for solving the CGVRP	travel for long distances which is why they must stop at AFSs. Thus, to solve the CGVRP, an algorithm needs to consider many factors like the number of AFVs, the allocation, and tank capacities. Exact algorithms cannot easily solve the large-sized strong-constraint problems, and heuristic algorithms do not guarantee the optimality of their solutions. So, meta-heuristic algorithms need to be developed to solve the problem.	are used. Through competitive search the traveling salesman problem (TSP) is further improved upon with local intensification. Customer adjustment and AFS adjustment operators are also applied. Finally, crossover operation generates the fittest solutions. Since multiple operators are implemented for solutions having different qualities, MAC is able to solve CGVRP effectively.	number of AFSs is 2. These instances are labelled as S series. For the group with large instances, the number of customers is set as 25, 50, 75, 100 and 150, and the number of AFSs is 2, 4, 6 and 8. The large instances are labelled as B series.		provide good solutions for any problem that can be generalized to the CGVRP It uses a competitive strategy that balances exploration and exploitation in the graphs, acting as a well-rounded heuristic for the CGVRP The MAC algorithm produced optimal solutions for every CGVRP instance it was operated on in very small computation times	that AFS, i.e., the original order of the graph is increased before it is operated upon	on which it was tested. The same values for the same instances were noted for the Ant Colony Optimization algorithm (ACO), the Discrete Firefly Algorithm (DFA) and the exact method by Gurobi. Not only did the MAC algorithm outperform all other methods in both instance sets, but it also produced the optimal solution for its best run on every instance considered in the paper. The literal values are presented in [13, Tab. 4] and [13, Tab. 5].

[14]	To find a solution to the VRP with the added constraint of minimization of fuel consumption and in turn CO_2 emissions.	To use Particle Swarm Optimization algorithm to optimize Vehicle Routing Plan. The objective function has been extended to consider fuel consumption minimization.	The data set is obtained by taking data from a particular day from a bottled drinking water company namely, Narmada Awet Muda, the largest distributor in Lombok Island, Indonesia. The data was taken for total product demand for each customer from Mataram city.	There are 3 major steps in the implementation of the method. Total fuel consumption: Calculating the total fuel consumption depending on the distance between the node, transportation speed, and the loading weight. Routing division: to create sub-routes based on requests from customers along with added constraints that contribute to forming the sub-route.	Particle Swarm Optimization Algorithm	It is a fitting algorithm to use where the variables are real numbers It has lesser calculations when compared to other heuristic optimization methods and mathematical techniques. It is suitable for solving VRP with complex combinatorial optimization problems. It has a relatively short computational time. It has the robustness to control parameters. The	This paper considers the algorithm only for a network with a single depot. All the customers cannot be assigned to a single sub-route due to time and load constraints.	The proposed route was compared with the initial route that was currently in use by the company. Upon multiple iterations of the PSO algorithm, we can observe a drop in fuel consumption which implies reduced fuel consumption. Hence, for fixed population parameters with population size N = 50, and fixed number of iterations = 200, we can obtain the best fitness function. it is seen that fuel consumption on

				Depending on the loading weight and maximum travel time constraint, the customer is added to the current sub-route else, the customer is assigned to the next sub route. Particle Swarm Optimization Algorithm: PSO is an optimization technique that mimics the behaviour of fish schooling. It contains a set of particles, and all particles move towards the optimal point. Implementation of the PSO		use of time-varying acceleration coefficients has led to increasing global search at the earliest time.		the proposed route is equivalent to 13% less than the fuel consumption of the initial route within the limited capacity and travel time.

				Algorithm with modification of the coefficient parameters according to Ratnaweera [15] who introduced time-varying acceleration coefficients to balance exploration and exploitation.				
[16]	To find a solution to the VRP with a focus on reducing carbon emissions and minimization of operational costs.	Implementation of a tuned Genetic Algorithm to find an appropriate solution to the GVRP.	The dataset has been taken from benchmarked instances in the CVRP library (available online).	There are 3 major steps in the implementation of the method. Conduct Literature Review: To understand the current status of research conducted in the domain. Mathematical Model: Implemented	Genetic Algorithm	The algorithm selects the shortest route with the mentioned constraints for the given parameters with the help of the fitness function which helps select the fittest population for reproduction. The algorithm improves over time providing	This paper assumes only a homogenous fleet of vehicles. This paper does not include factors such as road gradient into its calculations.	Analysis concludes that the optimum route with minimum cost was obtained when the tournament size was equal to 5, the generation size equal to 1000, with a mutation rate of 0.0025. The average percentage reduction in

				the mathematical model proposed by [17] with added modifications to include both the drivers' cost and maintenance cost into the overall operational costs while maintaining emission, capacity and route continuity constraints for a network with a single depot. Implementation and Parameter tuning of the GA: GAs are search-based algorithms inspired by natural selection.		more accurate results (or offspring).		cost obtained was 58.3%.

				The values for the size of population, replication, and generation are chosen along with mutation rates. The fitness function evaluates the percentage reduction in costs.				
[18]	To propose a method to find an optimum solution to the GVRP problem that aims to reduce greenhouse gas emissions per route.	Implementation of the Genetic Algorithm while incorporating elements of local and population search heuristics to minimize CO2 emissions per route on a set of light-duty diesel delivery vehicles.	Data for parcel deliveries in the Bristol area (UK) are used to create the real-world test instances.	There are 2 major steps in the implementation of the method. Create a model for emissions: A mathematical function created to calculate CO2 emissions over distance, speed, road gradient, load, emissions factor, and	Genetic Algorithm	The local search-based mutations provide a faster convergence of the GA Very high elitism leading to only better combinations or solutions forming a new population	This paper does not describe implementation pertaining to the use of asymmetric matrices to cater to the gradient method but rather considers a symmetric cost matrix of distance and speed models. Varying parameters affect runtime and best solutions differently.	The proposed solution resulted in a 17.91% reduction in distance and a 15.22% reduction in carbon emissions when compared with the original mTSP solution. It also performed better than the recorded route solution in both measures of

				constant coefficients. Formulate and implement the GA: The emissions model is used as the fitness function and some of the initial population is generated using heuristics while the rest is generated randomly. The population is kept small and methods like crossover are implemented using an ordered crossover approach while mutations are performed as a local search operator.				distance and emissions.

Detailed Analysis of the Chosen Problem and its Corresponding Algorithm

In this paper, the Mixed Integer Linear Programming approach is implemented to find the optimum solution for the Green Vehicle Routing Problem. It was one of the many algorithms that were analyzed in the Literature Review section.

The approach used comes under the Linear Programming domain of algorithms. Linear Programming is a method used for finding the optimum solution out of the available feasible solutions for a particular problem. This is done by either maximizing or minimizing the objective function depending on the use case and applying constraints to find the best solution. A linear programming problem is made up of two parts. The First part is the objective function which is a mathematical formulation of the primary objective of the problem. The second part is a system of linear equations which represents the constraints under which the optimum solution must be calculated.

Under Linear Programming depending on the value taken on by the decision variables it can be classified into different types of problems. In general decision variables are fractional to adopt a more realistic approach to the problem at hand. But depending on the problem sometimes fractional values are not suitable. This kind of problem is known as the Integer-programming problem. If some decision variables are allowed to take integer values and others fractional values that is if it comprises a mix of values this problem is known as the Mixed Integer Programming problem. And if the decision variables are restricted to only integer values it is known as the Pure Integer problem [11].

The main objective of the Vehicle Routing problem is to minimize the distance travelled by the vehicle and as well as cover each capital or node exactly once. Keeping the essence of the problem constant the objective of VRP was modified to suit the GVRP which is to minimize the emission rate of the vehicle. In this implementation, multiple vehicles (K) are considered wherein the start capital would be visited K times while all the other capitals would be visited exactly once by one of the vehicles.

The decision variables for this problem are:

- **x** which is a binary variable and takes on the value of 1 if a path exists between node i to node j otherwise it is set as 0.
- **u** which gives the order in which the nodes are visited

The constraint is that for every node excluding the start node the no of incoming edges or outgoing edges should be 1 whereas for the start node the number of incoming and outgoing edges should be equal to K.

Using these parameters and constraints the code was implemented for the GVRP problem [21].

Time Complexity of the implemented Algorithm

`import(libraries)`	# **O(1)**
`load(siteData)`	# **O(1)**
`load(positionData)`	# **O(1)**
`load(timeData)`	# **O(1)**
`load(distanceData)`	# **O(1)**
`speedData <- distanceData/timeData`	# **O(n^2)**
`emissionFactor <- f(speedData)`	# **O(n^2)**
`emissionData <- emissionFactor * distanceData`	# **O(n^2)**
`positions <- dictionary(positionData)`	# **O(n^2)**
`emissionDict <- dictionary(emissionData[node1][node2])`	# **O(n^2)**
`K <- k`	# **O(1)**
`startNode <- selected`	# **O(1)**

starting node	
problem <- creating the LP problem	# **O(1)** since it returns only name, and type of LP problem
x <- LP_Variable(emissionDict)	# **O(n^2)** since number of variables created under x = number of entries in emissionDict
u <- LP_Variable(siteData)	# **O(n^2)** since number of variables created under u = number of nodes (i.e. entries in siteData)
# creating the objective function cost <- LPSum(x*emissionDict)	# **O(n^2)** since we are creating an objective function equation summing the product of emissions with the corresponding node pair variable in x. Since there exist n^2 pairs, time complexity to access is O(n^2)
problem <- problem + cost	# **O(1)** adding the objective to the problem definition

# constraint definition 	
 	
for s in siteData:
 	
  if s <> startNode:
 	
    capacity <- 1
 	
  else:
 	
    capcity <- K
 	
  # adding inbound and outbound connection constraints to problem definition
	# **O(n)**
	
	# **O(1)**
	
	# **O(1)**
	
	# **O(1)**

problem <- problem + eqn(LPSum(x[Node1, s]) == capacity) problem <- problem + eqn(LPSum(x[s, Node1]) == capacity)	# **O(n)** # **O(n)** # **O(n)** since there are n sites or n nodes in total and only (n-1) variables that can connect to one selected node # The whole for loop - O(n)*[3*O(1) + 2*O(n)] => **O(n^2)**
N <- numberOfNodes/K	# **O(1)**

# subroute elimination, adding subroute linear constraints to problem 	
 	
for i in siteData:
 	
for j in siteData:
 	
if i <> j and (i <> startNode and j <> startNode) and (i,j) in x:
 	
problem <- problem + eqn(u[i] - u[j] <= N*(1-x[i,j]) - 1)
	# **O(n)**
	
	# **O(n)**
	
	# **O(1)**
	
	# **O(1)** since just adding to problem definition
	
	# The whole block of nested for loops - O(n) * O(n) * 2*O(1) => **O(n^2)**

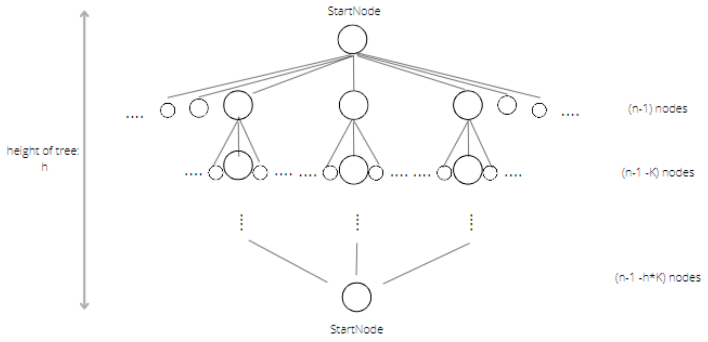
# solving the fomrulated problem	# **O(n^4)**

<table>
<tr>
<td>Solve(problem)</td>
<td>

'''

Explanation: The function implements CBC open source code library that makes use of a branch and cut algorithm to solve the mixed integer linear programming equation.

Assuming branch and cut for the following algorithm, we can visualize the problem as a tree.



At each level, K nodes get pruned due to branch and cut method, therefore at each level there exist (n-1-(x-1)*K) nodes to choose from for each K where x is the current level of tree. Height of the tree h = average number of nodes visited by each salesman => n/K.

Additionally, we have n^2 constraints to be solved to obtain the value of every variable (which tells us which next node to be picked creating an edge/branch).

Adding all together, we get the equation of time complexity to be:

$$n^2 \left[\, (n-1) + \sum_{x=1}^{h} K * [(n-1) - x * K] \,\right]$$

Upon solving and substituting for h as n/K, we get the time complexity:

O(n^4 + K*n^3) ; 1 <= K < n

As proven later in the results, the value of K has a big impact on the computation time. This

</td>
</tr>
</table>

	is because K is the coefficient of the next largest term in the complexity and thus this term cannot be ignored due to its heavy influence on computational time. '''
print Problem status: to know if optimal or not	# **O(1)**
non_zero_edges <- solutionEdges(x)	# **O(n^2)** since there are n^2 edges in x dictionary (all pairs of nodes)
tours <- get_next_site(startNode)	# **O(e)** where e is the number of solution edges (time complexity from function definition)
tours <- list([e] for e in tours)	# **O(K)** since it selects only K pairs (first pair from each list in tours) which denote the first node to visit by each of K salesman from startNode

for t in tours:
 	
while nextNode <> startNode:
 	
t.append((get_next_site (nextNode))[lastEle])
	# **O(K)** since tours has only K pairs
	
	# **O(n)** since there are (n-1) other nodes possible to be appended to t (depending on get_next site)
	
	# **O(e)**
	
	# The whole nested loop block takes O(K)*O(n)*O(e) => **O(K*n*e)**

for t in tours: print the firstNode for every node pair in t	# **O(K)** # **O(n/K)** since on average, t is a list of n/K elements # The whole block is of time complexity : O(K)*O(n/K) => **O(n)**

totalTime <- 0
 	
for t in tours:
 	
    time <- 0
 	
    for i:0 to noOfElements(t):
 	
        time <- time + timeData[t[i][0], t[i][1]]
 	
    if time > totalTime:
 	
        totalTime <- time
 	
print the totalTime
	# **O(1)**
	
	# **O(K)**
	
	# **O(1)**
	
	# **O(n/K)** since on average, t is a list of n/K elements
	
	# **O(1)** to access element by index
	
	# **O(1)**
	
	# **O(1)**
	
	# **O(1)**
	
	# The whole block is of time complexity : O(K)*[O(n/K) + 3*O(1)] => **O(n)**

''' Auxiliary function to get next edge: ''' def get_next_site(parent): edges = list(non_zero_edges given the fromNode in	# **O(e)** since there are e edges in non_zero_edges list

the pair is the parent node)	
for e in edges:	# **O(K)** since there are K selected first nodes from startNode
non_zero_edges.remove(e)	# **O(1)**
return edges	# **O(1)**
	# The whole function definition => O(e) + O(K)*O(1) => **O(e) since K < e**

Implementation

As seen in Figure 1. The first step is to import/install the necessary libraries required to run the program. The libraries are:

- **NumPy** – used to perform calculations with arrays.
- **Pandas** – used for data frame manipulation.
- **Matplotlib** – used for plotting graphs.
- **Seaborn** – used for data visualization.
- **PuLP** – used for solving optimization problems.

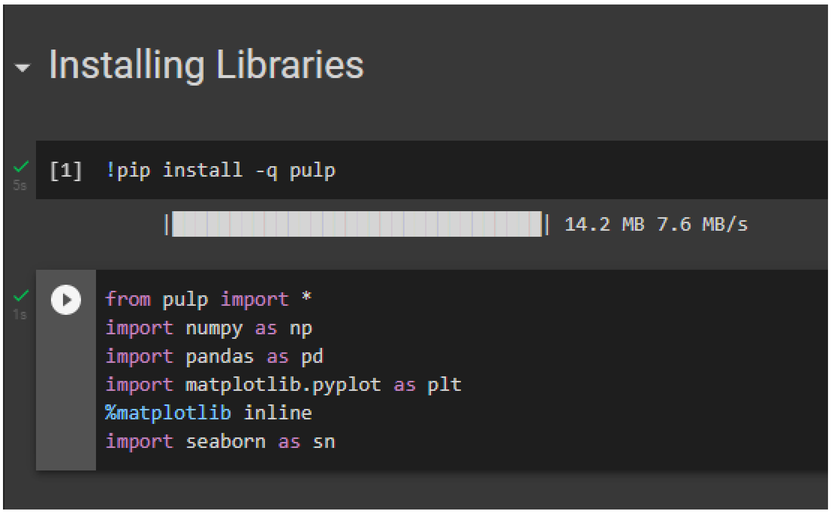


Figure 1. Importing/Installing Libraries

The next step is to load all the datasets as shown in Figure 2. The first file is “position.csv” which consists of the coordinates i.e., the longitude and latitude of each city. The next file “flight_time.csv” consists of a matrix of time taken from one node/city to every pair of nodes/cities. And finally, “distance.csv” contains a matrix of the distance from one node/city to every pair of nodes/cities. The 24 cities are given in the list **sites** and all these files are stored in variables **position**, **flighttime**, and **distance** using the pd.read_csv() function used to read comma-separated value files into the data frame using the pandas library.

Loading the Dataset

```
[4] sites = ['Barcelona','Belgrade','Berlin','Brussels','Bucharest','Budapest','Copenhagen','Dublin','Hamburg','Istanbul','Kiev',
          'London','Madrid','Milan','Moscow','Munich','Paris','Prague','Rome','Saint Petersburg','Sofia','Stockholm','Vienna','Warsaw']
    latlng = ['latitude', 'longitude']
    position = pd.read_csv('/content/position.csv', index_col="City")
    flighttime = pd.read_csv('/content/flight_time.csv', index_col="City")
    distance = pd.read_csv('/content/distance.csv', index_col="City")
```

Figure 2. Loading of Dataset

Next, the emissions matrix is calculated by first calculating the speed matrix which is done by dividing **distance/flighttime** and stored in variable **speed.** Then the null values are filled with zeroes in the speed matrix. The formula found in [22] is used to calculate the emissions as shown in Fig 3. This is then stored in the variable **emissions** which is a matrix that shows emissions of vehicles from one node/city to every pair of nodes/cities.

Forming Emissions Matrix

We calculate the cost of every edge as the amount of emissions output by travelling it:

> emissions = EF * d

where EF = 429.51 - 7.8227 x speed + 0.0617 x (speed^2)

```
[9] #Calculating speed matrix
    speed = distance/flighttime
    speed = speed.fillna(0)
```

```
[10] EF = 429.51 - 7.8227*speed + 0.0617*(speed**2)
     emissions = EF.mul(distance)
```

Figure 3. Calculation of Speed and Emission Matrix

The code for plotting the nodes is shown in Figure 4. The nodes are plotted by storing the coordinates for every city in **sites** with respect to all the other cities as key-value pairs in a dictionary called **positions.**

Plotting the Nodes

```
[12] positions = dict( ( city, (position.loc[city, 'longitude'], position.loc[city, 'latitude']) ) for city in sites)
```

```
for s in positions:
    p = positions[s]
    plt.plot(p[0],p[1],'o')
    plt.text(p[0]+.01,p[1],s,horizontalalignment='left',verticalalignment='center')

plt.gca().axis('off');
```

Figure 4. Plotting of Cities/Nodes

For the formulation of the problem section as shown in Figure 5 first, the emissions between two cities are stored in a dictionary called **distances** as key-value pairs. Next, the number of vehicles or the number of routes to be formed from the start node/city **K** is set as 3. The starting node is set as "Berlin" and the problem is formulated using the **LpProblem** class from the PuLP library. An instance of **LpProblem** is created and stored in **prob** and 2 parameters are passed to it. The first parameter is the name of the problem which in this case is 'vehicle' and the second parameter is used to define whether the problem is a maximization or minimization problem. This is a minimization problem since the emissions need to be reduced. Therefore, **LpMinimize** is passed.

The next cell is used to create the decision variables using **LpVariable** function. Wherein the first parameter "name" takes the name of the variable, the second parameter "indexes" takes a list of strings of the keys to the dictionary of LP variables, and the main part of the variable name itself, the third parameter "lowbound" takes the lower bound, fourth parameter "upBound" takes the upper bound and the last parameter "cat" takes the type of variable. Since this is a Mixed Integer programming problem the decision variables take on a mix of values.

Formulating the Problem

The problem is to minimise the emissions output for all routes formed. The number of routes to be formed is denoted by K.

```
#Get the distance between cities
distances = dict( ((s1,s2), emissions.loc[s1, s2] ) for s1 in positions for s2 in positions if s1!=s2)
```

```
#Setting the number of routes to be formed
K = 3
```

```
#Setting the starting point
starting_node = 'Berlin'
```

```
#Create the problem
prob = LpProblem("vehicle", LpMinimize)
```

```
#Variable to indicate if site i is connected to site j in the tour
x = LpVariable.dicts('x', distances, 0, 1, LpBinary)
#Dummy variables to eliminate subtours
u = LpVariable.dicts('u', sites, 0, len(sites)-1, LpInteger)
```

Figure 5. Formulation of the Problem using PuLP Library

The objective function is formulated by using the **LpSum** function from the PuLP library which allows the passing of a vector and calculates the sum of the list of linear expressions.

The vector passed here is the product of emissions multiplied with $\mathbf{x_{ij}}$ which is the decision variable used to indicate if a path is present from node **i** to node **j** for every **i,j** present in distances. This is stored in a variable called **cost** which is then added to the variable **prob**.

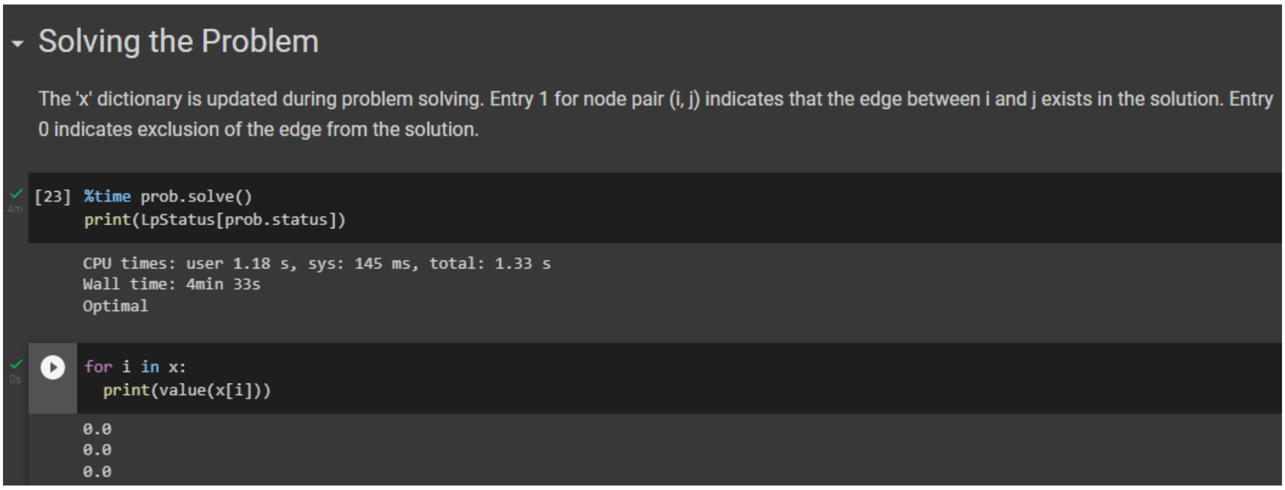
As shown in Figure 6 the constraints are added by again using the **LpSum** function. Here if **k** in **sites** is not equal to starting node which in this case is "Berlin" then **cap** is set to 1 otherwise it is set to **K** which is the no of times the starting node will be visited whereas all the other nodes will be visited exactly once. For all **i** in **sites** if a path is present from **i** to **k** then is added to prob and vice versa is also added to **prob.**

The sub tour elimination is done by first calculating **N** which is the total no of sites/ no of vehicles. For **i** and **j** in **sites**, if **i** is not equal to **j** and both **i** and **j** are not equal to starting node which is "Berlin" then for every **i, j in x** $\mathbf{u_i}$ **to** $\mathbf{u_j + N^* (1\text{-}x_{ij}) <=(N\text{-}1)}$ is formulated as constraints and added to **prob**.

```
[19] #Defining the objective and adding costs to the problem
     cost = lpSum([x[(i,j)]*distances[(i,j)] for (i,j) in distances])
     prob += cost

#Adding constraints
for k in sites:
    cap = 1 if k != starting_node else K
    #Defining number of inbound connections for each node
    prob += lpSum([x[(i,k)] for i in sites if (i,k) in x]) == cap
    #Defining number of outbound connections for each node
    prob += lpSum([x[(k,i)] for i in sites if (k,i) in x]) == cap

#Subtour elimination constraints
N = len(sites)/K
for i in sites:
    for j in sites:
        if i != j and (i != starting_node and j != starting_node) and (i,j) in x:
            prob += u[i] - u[j] <= (N)*(1-x[(i,j)]) - 1
```

Figure 6. Addition of Constraints

As shown in Figure 7 the problem is solved by calling a function **solve** which modifies the problem so that it is suitable for calculation and then the status of the solver is printed using the **LpStatus** function. During this process, every entry in the **x** dictionary is set to 1 if an edge exists between the corresponding pair of cities in the solution else it is set to 0.



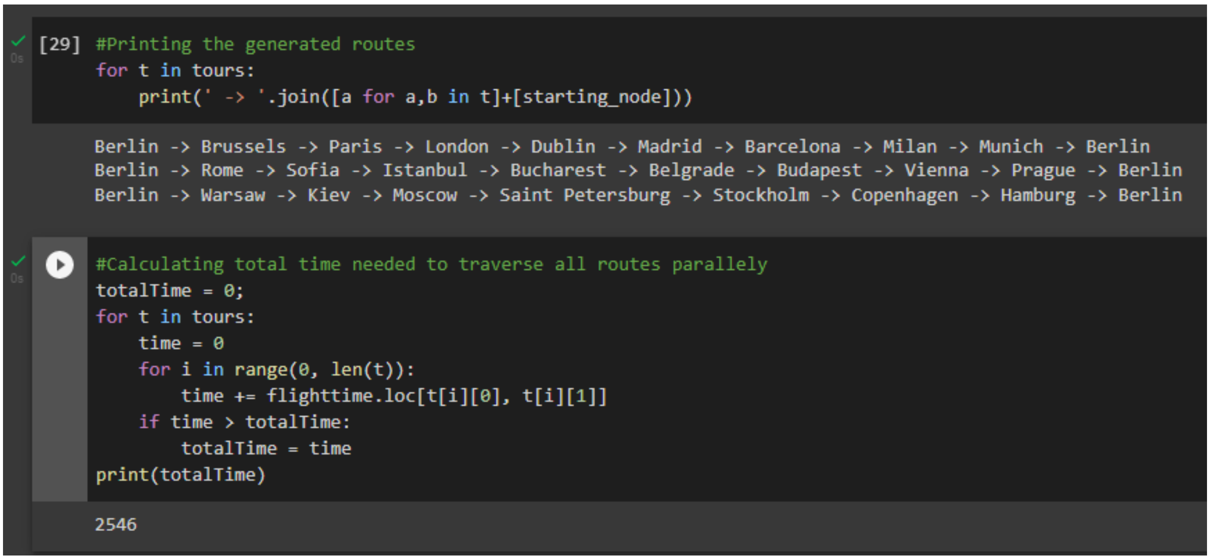
Solving the Problem

The 'x' dictionary is updated during problem solving. Entry 1 for node pair (i, j) indicates that the edge between i and j exists in the solution. Entry 0 indicates exclusion of the edge from the solution.

```
[23] %time prob.solve()
     print(LpStatus[prob.status])

     CPU times: user 1.18 s, sys: 145 ms, total: 1.33 s
     Wall time: 4min 33s
     Optimal

for i in x:
  print(value(x[i]))

0.0
0.0
0.0
```

Figure 7. Solving the Problem

The next section consists of converting the solution into a proper format. As shown in Figure 8 first, the non-zero edges are stored in variable **non_zero_edges**. The function **get_next_site** takes in an argument parent which is the start node in this case “Berlin”. And the routes which start from Berlin are stored in **edges** from the set of edges in **non_zero_edges**. Then for **e** in **edges** that is the **edges** present in **non_zero_edges** are removed and finally **edges** is returned and stored in a variable called **tours**. Then the following cells contain code that helps to create 3 paths since K=3 from the start node and corresponding routes from these three paths are stored in tours as a list of three lists.



```
[29] #Printing the generated routes
     for t in tours:
         print(' -> '.join([a for a,b in t]+[starting_node]))

     Berlin -> Brussels -> Paris -> London -> Dublin -> Madrid -> Barcelona -> Milan -> Munich -> Berlin
     Berlin -> Rome -> Sofia -> Istanbul -> Bucharest -> Belgrade -> Budapest -> Vienna -> Prague -> Berlin
     Berlin -> Warsaw -> Kiev -> Moscow -> Saint Petersburg -> Stockholm -> Copenhagen -> Hamburg -> Berlin

#Calculating total time needed to traverse all routes parallely
totalTime = 0;
for t in tours:
    time = 0
    for i in range(0, len(t)):
        time += flighttime.loc[t[i][0], t[i][1]]
    if time > totalTime:
        totalTime = time
print(totalTime)

2546
```

Figure 8. Translating the Solution

Figure 9 shows the 3 routes generated and the next cell is code corresponding to the calculation of the total time taken in minutes to visit all the nodes by the 3 vehicles simultaneously.

Translating the Solution

The solution is currently just a dictionary filled with 0s and 1s. This data needs to be translated into human readable routes.

```
[25] non_zero_edges = [ e for e in x if value(x[e]) != 0 ]

     def get_next_site(parent):
         #Helper function to get the next edge
         edges = [e for e in non_zero_edges if e[0] == parent]
         for e in edges:
             non_zero_edges.remove(e)
         return edges
```

```
[26] tours = get_next_site(starting_node)
```

```
[27] tours = [ [e] for e in tours ]
```

```
[28] for t in tours:
       while t[-1][1] != starting_node:
           t.append(get_next_site(t[-1][1])[-1])
```

Figure 9. Printing Routes and Calculation of Total Time

The final solution of the routes is then represented by plotting the three routes. Figure 10 shows the code for plotting the graph. And the output of the code is shown in Figure 11.

Final Solution

```
[31] #Plotting the tours
     colors = [np.random.rand(3) for i in range(len(tours))]
     for t,c in zip(tours,colors):
         for a,b in t:
             p1,p2 = positions[a], positions[b]
             plt.plot([p1[0],p2[0]],[p1[1],p2[1]], color=c)

     for s in positions:
         p = positions[s]
         plt.plot(p[0],p[1],'o')
         plt.text(p[0]+.01,p[1],s,horizontalalignment='left',verticalalignment='center')

     plt.title('%d '%K + 'people' if K > 1 else 'person')
     plt.xlabel('latitude')
     plt.ylabel('longitude')
     plt.show()
```

Figure 10. Code for plotting Final Solution

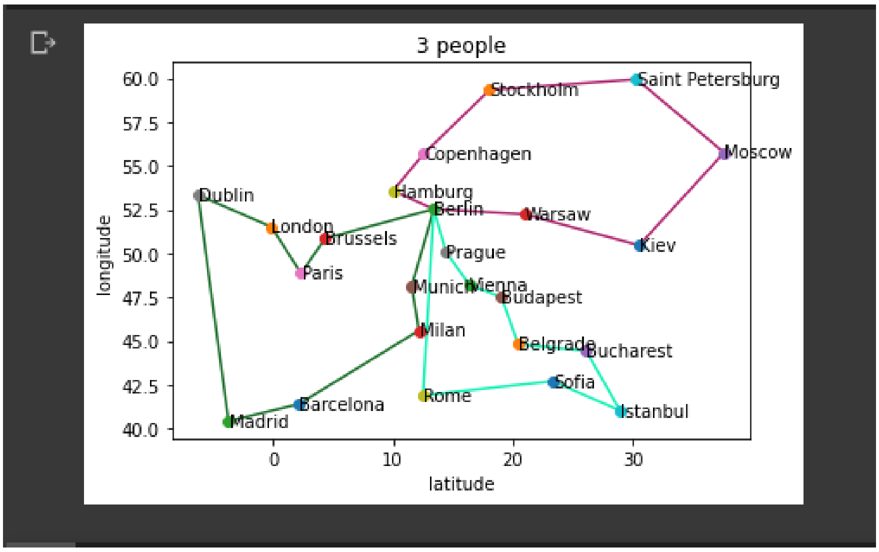


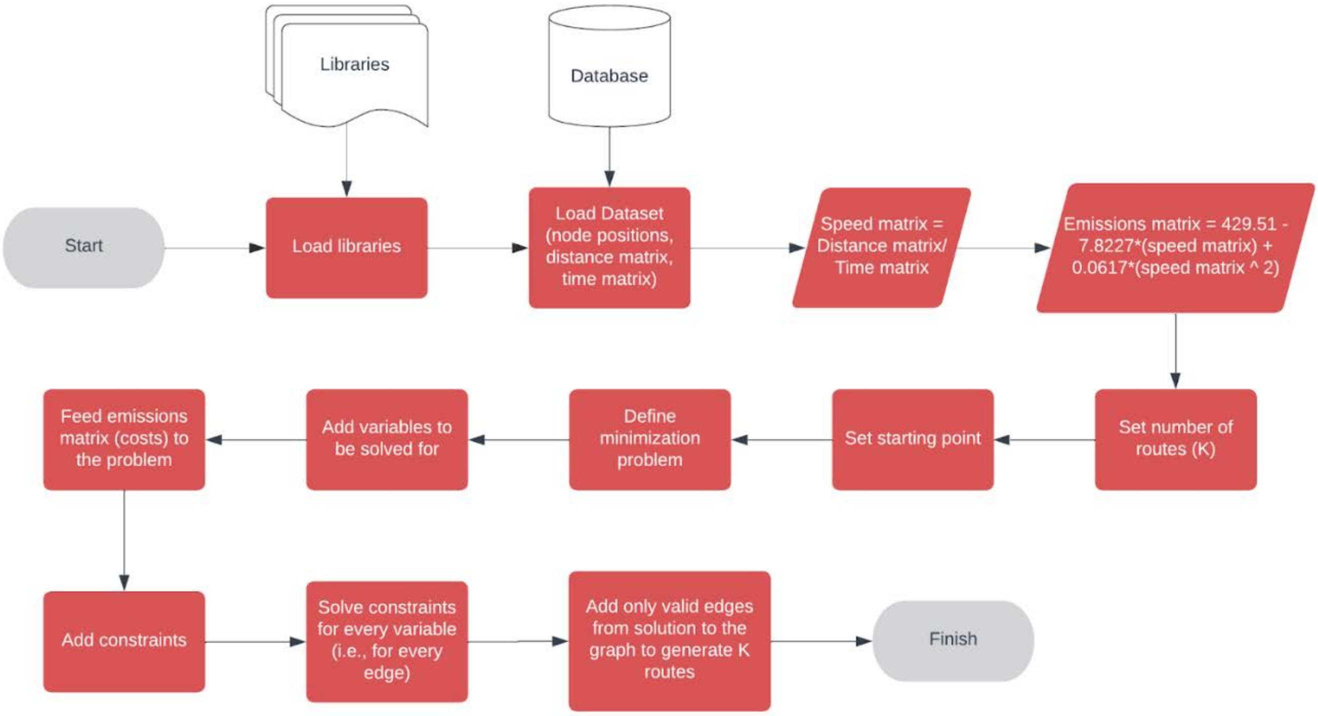
Figure 11. Plot of 3 Optimum Routes

In Figure 12 the time taken to traverse all the cities in minutes and the total emission during the traversal in kilograms are printed.

```
print('Longest time spent in route traversal:', totalTime, '(min)')
print('Total emissions during traversal:', value(prob.objective)/1000, '(kg)')

Longest time spent in route traversal: 2546 (min)
Total emissions during traversal: 5815.184188561756 (kg)
```

Figure 12. Printing of Time Taken for Traversal and Total Emissions

Architecture Diagram



Results and Discussion:

For the chosen dataset the positions of the nodes (relative to each other, based on their latitude and longitude) are shown in Figure 13:

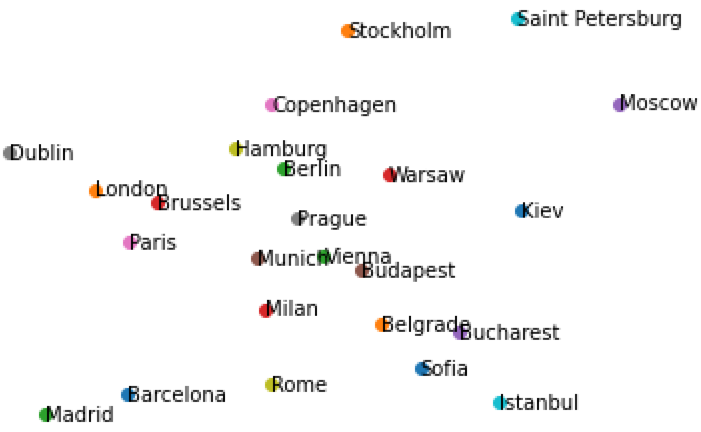


Figure 13. Positions of nodes in Dataset

1. Relation between the position of starting point and time taken to generate solution:

   For a given set of n nodes, we found that more time was needed to generate routes if the starting node was closer to the edge of the map than to the centre of it.

   Example:

   Let K = 2

   With Prague (closer to the centre of the map) as the starting point, the routes were generated in 6.36 s as shown in Figure 14.

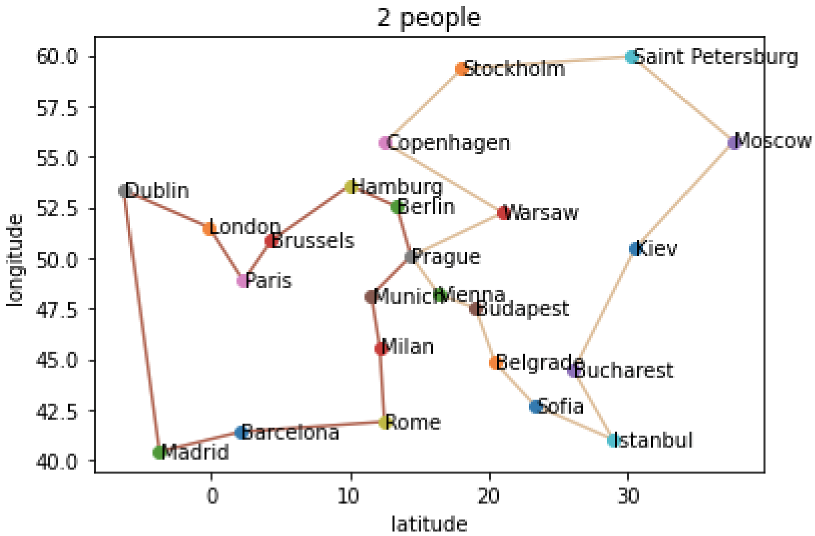


Figure 14. Prague as Starting Point

But with Budapest (farther from the center of the map) as the starting point, the routes were generated in 32.8 s as shown in Figure 15.

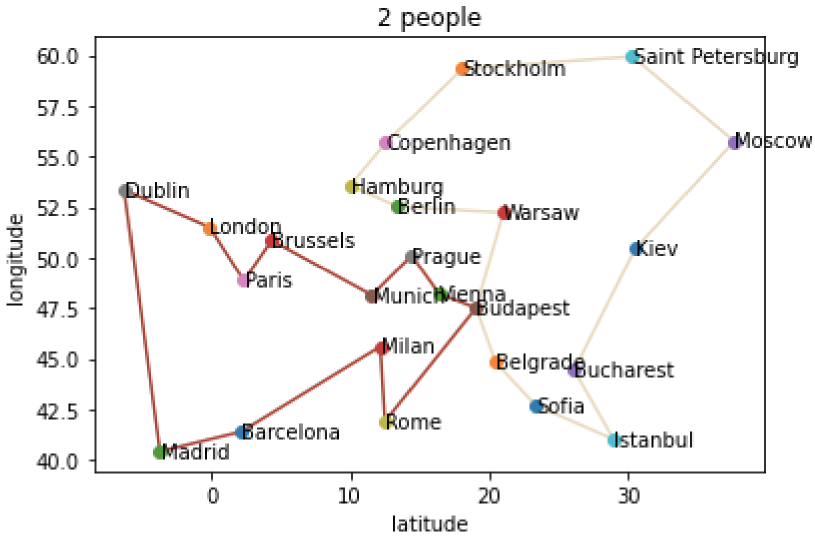


Figure 15. Budapest as Starting Point

2. Relation between the number of routes to be created and the time taken to generate the solution:

As proven while finding the time complexity of the algorithm, an increase in the number of routes to be created (K) leads to a significant increase in computation time.

Example:

Computation time with Berlin as starting node for K = 1, Time taken = 9.27 s as shown in Figure 16.

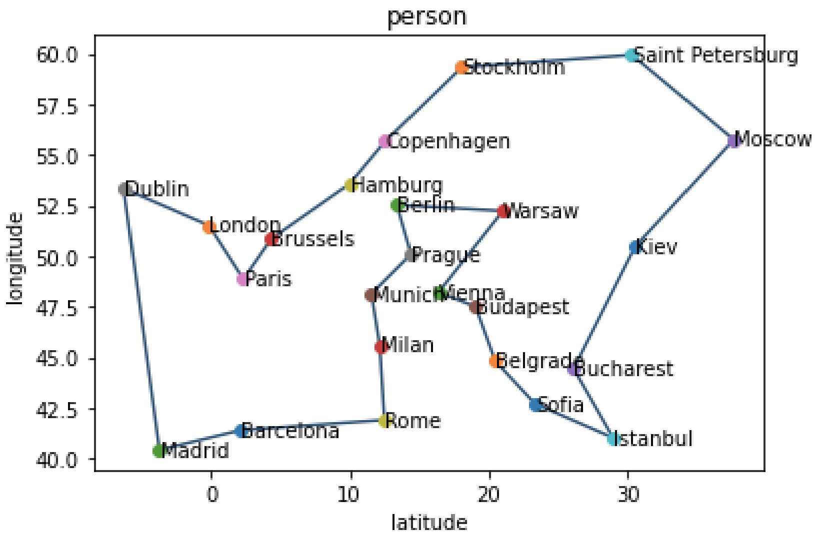


Figure 16. Berlin as Starting Point and K=1

K = 2 and Time taken = 22.24 s as shown in Figure 17.

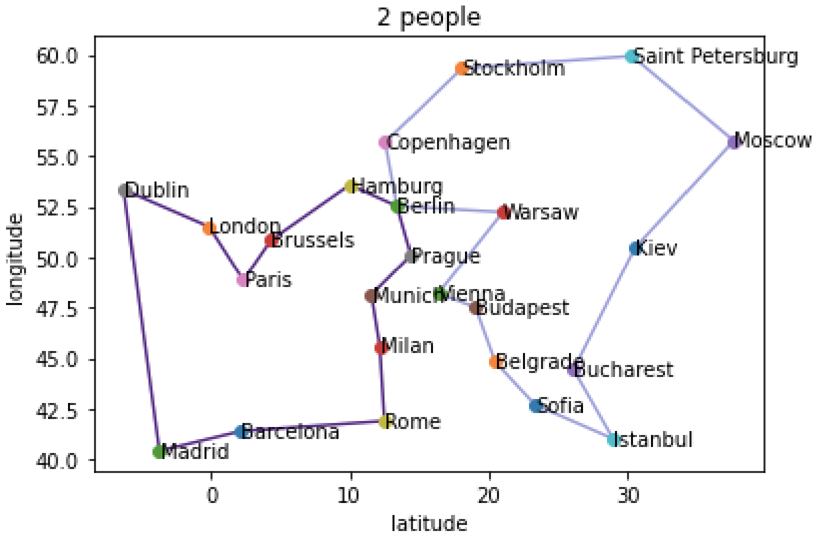


Figure 17. Berlin as Starting Point and K=2

K = 3 and Time taken = 4 min 57 s as shown in Figure 18.

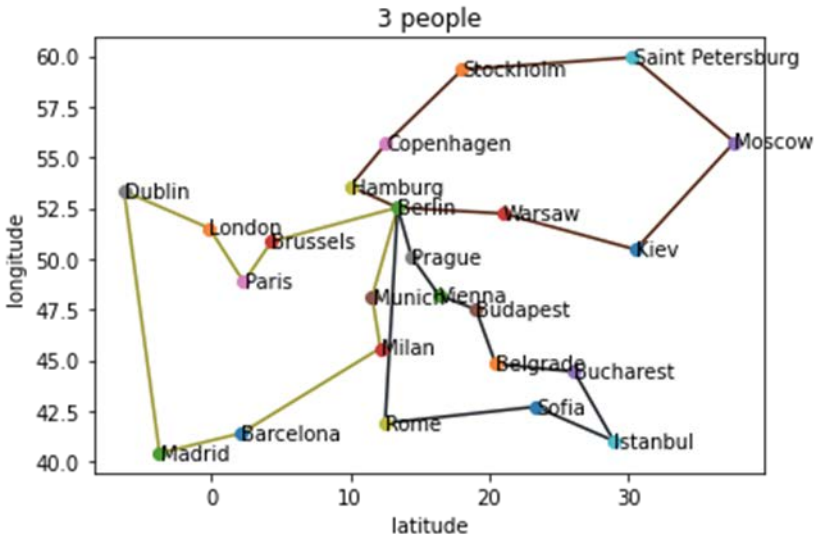


Figure 17. Berlin as Starting Point and K=3

3. Relation between the number of nodes and the time taken to generate the solution:

If 12 of the 24 nodes from the original dataset are removed, then the plot is as shown in Figure 18.

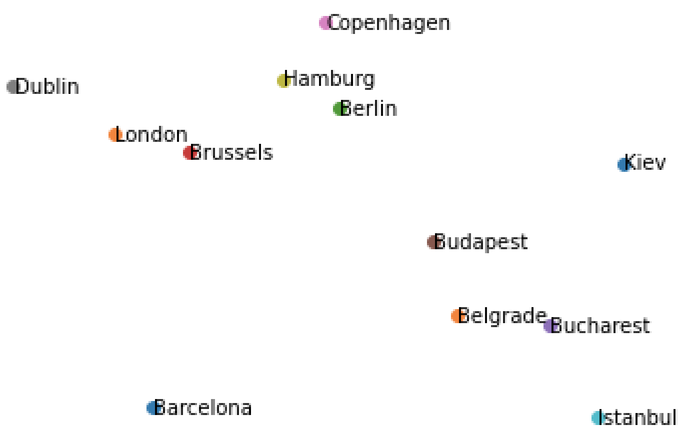


Figure 18. No of Nodes is 12

As seen previously, the time taken to generate K = 2 routes with Berlin as the starting point is 22.24 s. With the new plot, having half the original number of nodes, the time taken to do the same is 2.17 s as shown in Figure 19.

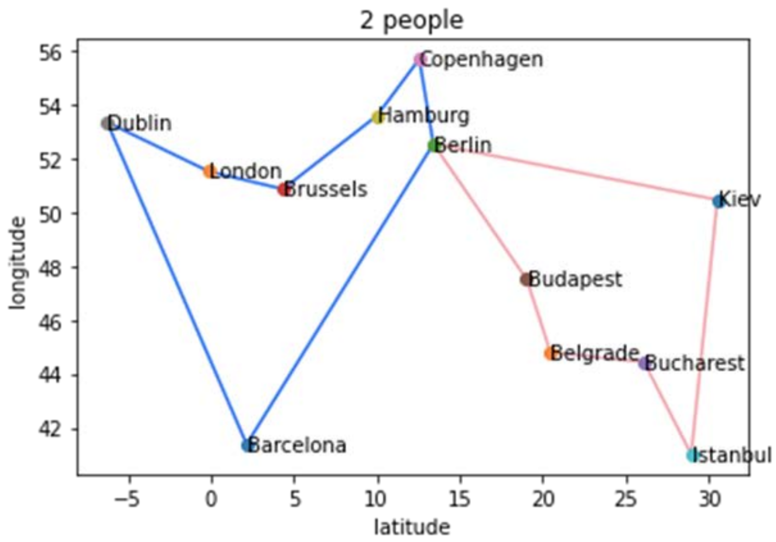


Figure 19. No of Nodes is 12, Starting Point is Berlin and K=2

Thus, computation time decreases with a decrease in the number of nodes (as proven in the time complexity analysis).

The paper begins by identifying and defining the Green Vehicle Routing (G-VRP) problem which aims at solving the VRP with environmental considerations to reduce emissions. We have focused on the Pollution Routing Problem (PRP), a subcategory of G-VRP. Phase-1 reviewed different proposed solutions and algorithms such as Dynamic Programming, Genetic Algorithm, Large Neighbourhood Search, Branch and fix algorithm, Particle Swarm Optimization, and path-based Mixed Integer Linear Programming (MILP) approach to solve the G-VRP. In Phase-2, we analyzed and implemented an algorithm that would provide an optimal and fitting solution. For the implementation part of the program, we selected the path-based MILP approach which aimed at reducing the NP-Hard problem (which generally takes at least exponential time) into a problem that can be readily solved in polynomial time by defining a minimizing linear equation. We adapted the general VRP problem that uses the branch and cut algorithm to solve the problem and modified it to take in emission considerations to fit the problem definition chosen. The scope of the implementation made use of a speed model to calculate the emissions factor per unit distance (according to MEET EU_2020 REPORT) [18] and used this model to identify the most efficient paths while minimizing CO_2 emissions.

With the above results in mind, it can be confidently said that linear programming works well for generating optimal solutions for GVRP. However, it is a time-consuming approach. Thus, linear programming is an efficient approach for small GVRP instances having greater clustering of nodes.

Through literature review and hands-on execution of the above algorithm, we can conclude that linear programming can work efficiently (with respect to computation time) on larger and sparser GVRP instances if used in conjunction with other programming approaches such as local search or metaheuristic algorithms. Given that it works very well for smaller instances, linear programming could perhaps be used as a subproblem solver to generate solutions for larger instances (in a divide and conquer approach) in future works.

Further research needs to be conducted to modify the following optimal algorithm to include factors such as load, road gradient, type and efficiency of a vehicle, and other factors to make the emissions model more accurate. Additionally, the future scope includes considering other constraints under the G-VRP such as multiple pick-ups and drop-offs, multi-depot considerations, time windows, etc. to create an optimal and robust algorithm to suit various needs and situations efficiently, and accurately.

Conclusions

Upon summing the Big O for every operation in the above code, we can conclude that the largest factor that affects the time complexity (Big O) of the code is the function that finds the solution to the Linear programming problem and takes **O(n^4)** time**.** Also, we have managed to reduce the complexity of the GVRP problem from exponential to polynomial time complexity.

References:

[1] M. Soysal, M. Çimen, Ç. Sel, and S. Belbağ, “A heuristic approach for green vehicle routing,” RAIRO - Operations Research, vol. 55, pp. S2543–S2560, 2021, doi: 10.1051/ro/2020109.

[2] J. Dutta et al., “MULTI-OBJECTIVE GREEN MIXED VEHICLE ROUTING PROBLEM UNDER ROUGH ENVIRONMENT,” Transport, vol. 0, no. 0, pp. 1–13, Feb. 2021, doi: 10.3846/transport.2021.14464.

[3] NEO, “Capacitated VRP Instances,” Networking and Emerging Optimization (NEO), 2013. https://neo.lcc.uma.es/vrp/vrp-instances/capacitatedvrp-instances (accessed Feb. 22, 2022).

[4] T. Ralphs, “Vehicle Routing Data Sets,” Oct. 03, 2003. https://www. coin-or.org/SYMPHONY/branchandcut/VRP/data/index. htm.old (accessed Feb. 22, 2022).

[5] VRP-REP, “Datasets,” Vehicle Routing Problem Repository (VRP-REP), 2018. http://www.vrp-rep.org/datasets.html (accessed Feb. 22, 2022).

[6] P. Augerat, D. Naddef, J. M. Belenguer, E. Benavent, A. Corberan, and G. Rinaldi, “Computational results with a branch and cut code for the capacitated vehicle routing problem,” www.osti.gov, Sep. 1995, Accessed: Feb. 22, 2022. [Online]. Available: https://www.osti.gov/etdeweb/biblio/289002.

[7] P. Barma, “ZIP Archive: Rough Data,” 2018. https://drive.google.com/file/d/1W8pJvibW3v3mXrpz_ 9u0qjmvd_yLWecQ/view (accessed Feb. 22, 2022).

[8] R. Eshtehadi, M. Fathian, S. Pishvaee, and E. Demir, “A hybrid metaheuristic algorithm for the robust pollution-routing problem.” Accessed: Feb. 27, 2022. [Online]. Available: http://www.jise.ir/article_53645_309bf1167d1cc0e2366fcb527f6b06ca.pdf.

[9] “The Pollution-Routing Problem Instance Library.” http://www.apollo.management.soton.ac.uk/prplib.htm (accessed Feb. 27, 2022).

[10] M. Murua, D. Galar, and R. Santana, “Solving the multi-objective Hamiltonian cycle problem using a Branch-and-Fix based algorithm,” Journal of Computational Science, vol. 60, p. 101578, Apr. 2022, doi: 10.1016/j.jocs.2022.101578.

[11] M. Bruglieri, S. Mancini, F. Pezzella, and O. Pisacane, “A path-based solution approach for the Green Vehicle Routing Problem,” Computers & Operations Research, vol. 103, pp. 109–122, Mar. 2019, doi: 10.1016/j.cor.2018.10.019.

[12] J. Andelmin and E. Bartolini, “A multi-start local search heuristic for the Green Vehicle Routing Problem based on a multigraph reformulation,” Computers & Operations Research, vol. 109, pp. 43–63, Sep. 2019, doi: 10.1016/j.cor.2019.04.018.

[13] L. Wang and J. Lu, “A memetic algorithm with competition for the capacitated green vehicle routing problem,” IEEE/CAA Journal of Automatica Sinica, vol. 6, no. 2, pp. 516–526, Mar. 2019, doi: 10.1109/jas.2019.1911405.

[14] B. N. I. F. Ramadhani and A. K. Garside, “Particle Swarm Optimization Algorithm to Solve Vehicle Routing Problem with Fuel Consumption Minimization,” Jurnal Optimasi Sistem Industri, vol. 20, no. 1, pp. 1–10, May 2021, doi: 10.25077/josi.v20.n1.p1-10.2021.

[15] A. Ratnaweera, S. K. Halgamuge, and H. C. Watson, "Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients,” IEEE Transactions on evolutionary computation, vol. 8, no. 3, pp. 240-255, 2004. doi: 10.1109/TEVC.2004.826071

[16] E. Waidyathilaka, V. Tharaka, and R. Wickramarachchi, “Multi-objective Green Vehicle Routing Optimization,” 2020. Accessed: Feb. 27, 2022. [Online]. Available: http://www.ieomsociety.org/ieom2020/papers/152.pdf.

[17] E., J., Panickera, V. V. & Sridharan, R., 2015. Multi-objective optimization model for a green vehicle routing problem. Procedia - Social and Behavioral Sciences, Volume 189, pp. 33-39.

[18] P. R. de Oliveira da Costa, S. Mauceri, P. Carroll, and F. Pallonetto, “A Genetic Algorithm for a Green Vehicle Routing Problem,” Electronic Notes in Discrete Mathematics, vol. 64, pp. 65–74, Feb. 2018, doi: 10.1016/j.endm.2018.01.008.

[19] S. Erdoğan and E. Miller-Hooks, “A Green Vehicle Routing Problem,” Transportation Research Part E: Logistics and Transportation Review, vol. 48, no. 1, pp. 100–114, Jan. 2012, doi: 10.1016/j.tre.2011.08.001.

[20] J. Andelmin and E. Bartolini, “An Exact Algorithm for the Green Vehicle Routing Problem,” Transportation Science, vol. 51, no. 4, pp. 1288–1303, Nov. 2017, doi: 10.1287/trsc.2016.0734.

[21] S. Anbuudayasankar, K. Ganesh, and K. Mohandas, “Mixed-Integer Linear Programming for Vehicle Routing Problem with Simultaneous Delivery and Pick- Up with Maximum Route-Length,” The International Journal of Applied Management and Technology, vol. 6, no. 1, [Online]. Available: https://scholarworks.waldenu.edu/cgi/viewcontent.cgi?article=1020&context=ijamt

[22] P. R. de Oliveira da Costa, S. Mauceri, P. Carroll, and F. Pallonetto, “A Genetic Algorithm for a Green Vehicle Routing Problem,” Electronic Notes in Discrete Mathematics, vol. 64, pp. 65–74, Feb. 2018, doi: 10.1016/j.endm.2018.01.008.