UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF MECHANICAL ENGINEERING

The Period Vehicle Routing Problem: Modeling and Solution

by

XIROU AIKATERINI

Submitted in partial fulfillment of the requirements for the degree of Diploma in Mechanical Engineering at the University of Thessaly

Volos, 2025

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF MECHANICAL ENGINEERING

The Period Vehicle Routing Problem: Modeling and Solution

by

XIROU AIKATERINI

Submitted in partial fulfillment of the requirements for the degree of Diploma
in Mechanical Engineering at the University of Thessaly

Volos, 2025

Approved by the Committee on Final Examination:

Advisor	Dr. George Saharidis, Associate Professor, Department of Mechanical Engineering, University of Thessaly
Member	Dr. Dimitrios Pandelis, Professor, Department of Mechanical Engineering, Aristotle University of Thessaly
Member	Dr. George Kozanidis , Professor, Department of Mechanical Engineering, University of Thessaly

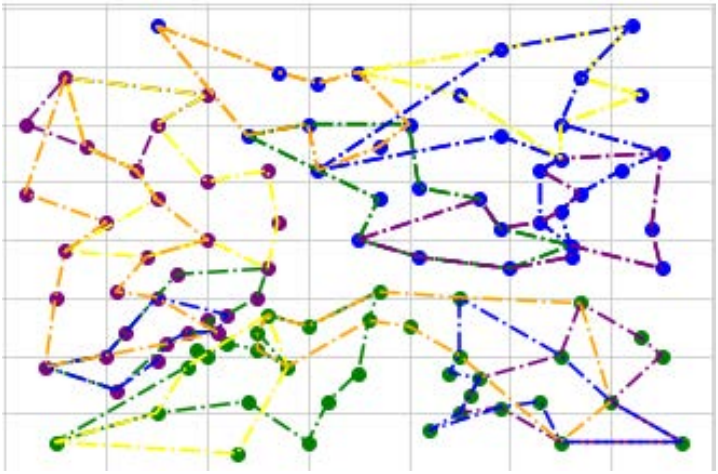
Date Approved: [27/06/2025]

The Period Vehicle Routing Problem: Modeling and Solution

XIROU AIKATERINI

Department of Mechanical Engineering, University of Thessaly

Supervisor: Dr George Saharidis
Associate Professor of Operational Research



ABSTRACT

In this thesis is introduced the **V**ehicle **R**outing **P**roblem (**VRP**) and is examined the generalization of this, which is the **P**eriod **V**ehicle **R**outing **P**roblem (**PVRP**). The purpose of the VRP is applying points in routes in a day starting and ending in a single depot. The routes of the generalization of the problem must be constructed over multiple days. According to several solution methods the PVRP is separated into several VRPs to satisfy the requirements of points for more than one visit per period. The main purpose is to find a set of tours for each vehicle that minimizes the total cost or the total time or the total distance while satisfying operational constraints. It is presented the first formulation and the evolution of the model and some variants of it, such as PVRP with Time Windows (RVRPTW), multi-depot (MDPVRP) and PVRP with service choice (PVRPSC). Moreover, it is presented the solution methods for the problems and for the variants that researchers used, which are the classical heuristic, the metaheuristic, and the programming approach. Some applications that are used in our days are small package shipping, automated meter reading (AMR) with radio frequency identification (RFID) technology and container terminal operations. Moreover, an algorithm for solving the 100-point problem of the N. CRISTOFIDES AND J.E.BEASSLEY paper is described, which aims to find the optimal routes for 3 salesmen within the 5-day period. Specifically, customers are represented by these 100 points,

with each requiring a different frequency of visits (up to 3 visits per 5 days). The algorithm consists of 5 steps which are briefly explained below:

Step 1: We calculate the center of gravity of the clients' points

Step 2: We divide the points into 3 regions, because we have 3 salesmen, so that they are evenly distributed according to their geographical distribution

Step 3: We divide the points into 3 regions, so that the turnover/load disbursed by each seller from the points in his region is evenly distributed

Step 4: This step is triggered when the application of steps 2 and 3 results in overlaps in the areas of the salesmen and aims to smooth out possible incorrect assignments of points to a salesman who is far from his coverage area.

As there is a possibility that this step is not satisfied, there is the possibility to choose to satisfy the equal distribution of points based on their spatial distribution, i.e. we choose the separation into areas that we did in step 2 or based on turnover, i.e. the separation into areas that we did in step 3.

Step 5: After we have divided the points into 3 responsibility areas, one for each seller, we create multiple virtual points in those that initially required more than one visit, so that all of our points (real and virtual) require one visit. We divide each vendor's region into 5 sub-regions requiring points with the same coordinates to be placed in different clusters. At the end each of these clusters will represent one day of the week, to which we apply a well-known TSP formulation in the literature to derive the optimal paths for each salesman of each day.

Key Words: VRP, CVRP, PVRP, TSP, Mixed-Integer Programming, Linear Programming, Non-Linear Programming, Heuristics, Metaheuristics, Python, Gurobi

ΠΕΡΙΛΗΨΗ

Στην παρούσα διατριβή παρουσιάζεται το πρόβλημα δρομολόγησης στόλου οχημάτων (VRP) και εξετάζεται η γενίκευσή του, το οποίο είναι το πρόβλημα δρομολόγησης στόλου οχημάτων με περιοδικότητα (PVRP). Σκοπός του VRP είναι η δημιουργία ανάθεση σημείων σε διαδρομές σε μια ημέρα, οι οποίες θα ξεκινούν και θα καταλήγουν σε μια αποθήκη. Οι διαδρομές του γενικευμένου προβλήματος πρέπει να κατασκευαστούν για πολλές ημέρες. Σύμφωνα με διάφορες μεθόδους επίλυσης το PVRP διαχωρίζεται σε διάφορα VRP για να ικανοποιήσει τις απαιτήσεις των σημείων για περισσότερες από μία επισκέψεις ανά περίοδο. Ο κύριος σκοπός είναι η εύρεση ενός συνόλου διαδρομών για κάθε όχημα που ελαχιστοποιεί το συνολικό κόστος, τον συνολικό χρόνο ή τη συνολική απόσταση, ικανοποιώντας παράλληλα τους λειτουργικούς περιορισμούς. Παρουσιάζεται η πρώτη διατύπωση και η εξέλιξη του μοντέλου και ορισμένες παραλλαγές του, όπως το PVRP με χρονικά παράθυρα (RVRPTW), το πρόβλημα των πολλαπλών αποθηκών (MDPVRP) και το PVRP με επιλογή υπηρεσιών (PVRPSC). Επιπλέον, παρουσιάζονται οι μέθοδοι επίλυσης των προβλημάτων και των παραλλαγών που χρησιμοποίησαν οι ερευνητές, οι οποίες είναι η κλασική ευρετική, η μεταευρετική και η προγραμματιστική προσέγγιση. Ορισμένες εφαρμογές που χρησιμοποιούνται στις μέρες μας είναι η αποστολή μικρών πακέτων, η αυτόματη ανάγνωση μετρητών (AMR) με την τεχνολογία αναγνώρισης ραδιοσυχνοτήτων (RFID) και η λειτουργία τερματικών σταθμών εμπορευματοκιβωτίων.

Επιπλέον, περιγράφεται ένας αλγόριθμος για την επίλυση του προβλήματος των 100 σημείων της εργασίας των N. CRISTOFIDES AND J.E.BEASSLEY, ο οποίος αποσκοπεί στην εύρεση των βέλτιστων δρομολογίων για 3 πωλητές εντός περιόδου 5 ημερών. Συγκεκριμένα, οι πελάτες αντιπροσωπεύονται από αυτά τα 100 σημεία, με το καθένα να απαιτεί διαφορετική συχνότητα επισκέψεων (έως 3 επισκέψεις ανά 5 ημέρες). Ο αλγόριθμος αποτελείται από 5 βήματα τα οποία εξηγούνται συνοπτικά παρακάτω:

Βήμα 1: Υπολογίζουμε το κέντρο βάρους των σημείων των πελατών

Βήμα 2: Χωρίζουμε τα σημεία σε 3 περιοχές, επειδή έχουμε 3 πωλητές, έτσι ώστε να είναι ομοιόμορφα κατανεμημένα σύμφωνα με τη γεωγραφική τους κατανομή

Βήμα 3: Χωρίζουμε τα σημεία σε 3 περιοχές, έτσι ώστε ο κύκλος εργασιών/φορτίο που εκταμιεύει κάθε πωλητής από τα σημεία της περιοχής του να κατανέμεται ομοιόμορφα

Βήμα 4: Το βήμα αυτό ενεργοποιείται όταν η εφαρμογή των βημάτων 2 και 3 οδηγεί σε επικαλύψεις στις περιοχές των πωλητών και αποσκοπεί στην εξομάλυνση πιθανών εσφαλμένων αναθέσεων σημείων σε έναν πωλητή που βρίσκεται μακριά από την περιοχή κάλυψής του.

Καθώς υπάρχει η πιθανότητα να μην ικανοποιηθεί το βήμα αυτό, υπάρχει η δυνατότητα να επιλέξουμε να ικανοποιήσουμε την ισοκατανομή των σημείων με βάση τη χωρική τους κατανομή, δηλαδή επιλέγουμε το διαχωρισμό σε περιοχές που κάναμε στο βήμα 2 ή με βάση τον τζίρο, δηλαδή το διαχωρισμό σε περιοχές που κάναμε στο βήμα 3.

Βήμα 5: Αφού χωρίσουμε τα σημεία σε 3 περιοχές ευθύνης, μία για κάθε πωλητή, δημιουργούμε πολλαπλά εικονικά σημεία σε εκείνα που αρχικά απαιτούσαν περισσότερες από μία επισκέψεις, έτσι ώστε όλα τα σημεία μας (πραγματικά και εικονικά) να απαιτούν μία επίσκεψη. Χωρίζουμε την περιοχή κάθε πωλητή σε 5 υποπεριοχές απαιτώντας τα σημεία με τις ίδιες συντεταγμένες να

τοποθετηθούν σε διαφορετικές υποπεριοχές. Στο τέλος κάθε μία από αυτές τις υποπεριοχές θα αντιπροσωπεύει μία ημέρα της περιόδου, στην οποία εφαρμόζουμε μία γνωστή διατύπωση του προβλήματος του πλανόδιου πωλητή (TSP) από τη βιβλιογραφία για να εξάγουμε τις βέλτιστες διαδρομές για κάθε πωλητή κάθε ημέρας.

Λέξεις-κλειδιά: VRP, CVRP, PVRP, TSP, ευρετικές μέθοδοι, μεταευρετικές μέθοδοι, μεικτός ακέραιος προγραμματισμός, γραμμικός και μη γραμμικός προγραμματισμός, Python, Gurobi

CONTENTS

LIST OF FIGURES

LIST OF TAPLES

LIST OF FORMULATIONS

1 INTRODUCTION

Vehicle routing problems define a class of combinatorial optimization problems that allow optimizing itineraries of a fleet of vehicles, when these vehicles operate round trips. We all connect with this kind of problem somehow in our life through city logistics, which is the process that optimize the transport activities in urban areas while considering the traffic environment, the traffic congestion and energy consumption. In our days, when the European Commission announce the goal of zero pollution of air, water and soil until 2050, is more relevant than ever the development of new algorithms and techniques to improve the personal and professional transport.

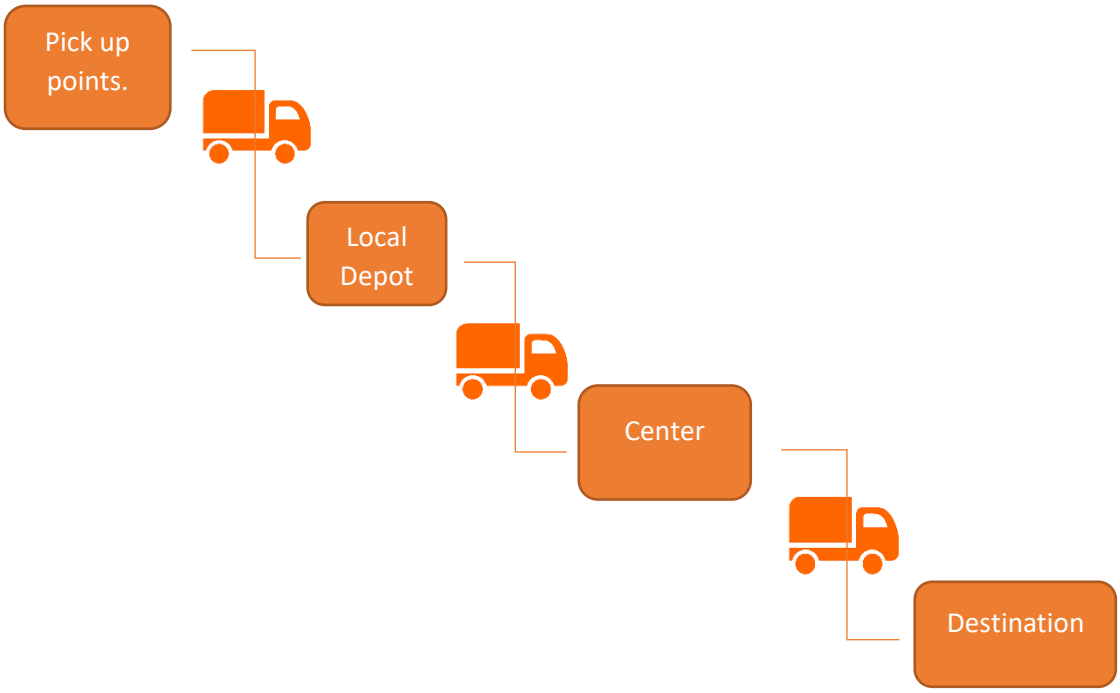
The PVRP was introduced in the seminal paper by Beltrami and Bodin [3] in 1974 for assigning hoist compactor trucks in municipal waste collection where the garbage sites need to be visited with different frequencies. The authors allow two schedules MWF (Monday- Wednesday-Friday visits) and TRS (Tuesday-Thursday-Saturday), each node and its copy are assigned to different schedule so only two VRP's need to be solved. Russell and Igo [4] provide a formal definition of the PVRP as a mixed integer problem, the "Assignment Routing Problem" and examine the difficulties of choosing a schedule for each node. The authors did not formulate the problem or attempt to solve it optimally but propose three heuristics instead. Christofides and Beasley [5] presented the first integer programming formulation of the PVRP using two sets of decision variables, one for the assignment of customers to schedules and another for the routing of a given vehicle on a given day. The formulation follows the VRP formulation of Golden et al [6] with three sets of decision variables.
Tan and Beasley [7] summarize the results of Beltrami and Bodin, Russell and Igo and Christofides and Beasley and propose a formulation with a cost measure. Given the difficulty of solving the problem, the authors made the decision of allocating nodes to days in the first phase and the routing decision for each day in the second phase.

PVRPs are part of supply chain and have evolved into a significant body of work with several exciting variants and applications arising in recent years such as supermarket chain, recycling, waste collection, milk collection problems arising in the dairy industry, general logistics and collecting recyclable material. In section 2 is presented some applications for the problem.

2 APPLICATIONS

2.1 SMALL PACKAGE SHIPPING

One application with vital meaning and commercial revenues is small package shipping. It's a national and international case which operates at maximum efficiency.



In the diagram above the process of this application is described. First the packages are picked up from branch store locations and then be brought to a local depot. After that they are transported through the national or international network, to centers that are in an appropriate distance from the final location / destination. The vehicles that are associated with a center handle both the local pickup and delivery tasks. The fleet management issues are principal focuses of research and development effort.

Small packages pickup and delivery service has several characteristics that are in common and some that distinguish it from the VRP and PVRP model. In classical VRP approach the minimization of travel cost is founded by optimizing routings for each vehicle. The first and the last point is a central depot, and each point has an amount of demand that must be satisfied without exceeding the vehicle's capacity. This problem is described in detail in the following chapter, in section 3.1.

Some of the factors that should be part of the planning and not be skipped are the following:

- Point Schedule Consistency: in the small package service the points want to be serviced with the same service provider (SP) at about the same time over multiple days. This may increase the cost of a route. The VRP is a static model so this factor is not an issue and the PVRP model handles a large number (perhaps over 700) of points so the truck driver or the service time is not a debate but a planning issue.

- Service Territory Management: in the small package service customers want to be familiar with the SP so they wish to be serviced by the same SP every day. The VRP is a static model so the service territories are fixed. The PVRP is a dynamic model with multiple day planning horizons, so service territories or neighborhoods are decision factors.

- Package Selection Time: is the time required for the SP to find the appropriate package in the rear of the vehicle. On most of the vehicles there are two or more (in depend on the capacity of the vehicle) sets of shelves running along each side of it where packages can be arranged. The order in which the packages are loaded on the shelves is made so that this time factor is minimized. This factor should be considered in the small package services, VRP, PVRP and in the big family of logistic applications.

- Time Windows: When a customer or point asks for a time window means that there is a need for flexibility in arrival time or departure time, so in small package service business is identified as an extra paid service. In this case, because of the small number of areas, there is flexibility to change the plan in the day so customers can ask for a time window even in the same day, but in VRP and PVRP there is no such opportunity. VRP and PVRP with time windows are variants of the original problems and have separate solution methods.

- Unplanned Events: such as traffic congestion or road construction or an accident can push the SP to make a new plan for the deliveries. In VRP and PVRP this can happen under the danger not only increase the distance and the cost of the routing but also not serving some points or losing time windows.

2.2 AMR WITH RFID TECHNOLOGY

One of the most popular applications in the US is the use of automated meter reading (AMR) with radio frequency identification (RFID) technology, which allows utility companies to read utility meters from a distance. Therefore, utility companies do not need to visit every customer/point. This changes the problem of monthly metering gas, water, or electricity usage from a traveling salesman problem (TSP) to a close enough traveling salesman problem (CETSP), over a realistic street network. The use of RFID technology has increased, and it is utilized extensively in many industries for the tracking of resources. The accuracy of transmitters and receivers has improved, and the cost has decreased. Therefore, the viability and usefulness of these devices has increased.

First, RFID refers to as Automatic Identification and Data Capture (AIDC), which are methods automatically identify objects, collect data about them, and enter those data directly into computer systems with little or no human intervention. Digital data encoded in RFID tags are captured by a reader via Dario waves. This technology is the next generation of barcoding in the data from a tag, which is captured by a device that stores the data in database. An important advantage of RFID over barcoding is that RFID tag data can be read outside the line-of-sight, whereas barcodes must be aligned with an optimal scanner. RFID tags contain an integrated circuit and an antenna, the reader then converts the radio waves to a more usable form of data. Information collected from the tags is then transferred through a communications interface to a host computer system, where the data can be stored in a database and analyzed later.

Second, utility companies provide natural gas, electricity, water, and other necessities for both residential and commercial properties. These make profit but are typically heavily regulated by public authorities. The increased cost of salaries and benefits of meter readers coupled with the rise of gasoline prices has become a financial problem. These companies are interested in monitoring customers demand and usage, so they can control production and maximize efficiency. One way that utilities can both improve their customer service and reduce their meter reading cost is to focus their efforts on AMR coupled with RFID technology.

There are two cases to consider. In one case, the meter sends a signal to a stationary receiver that is located nearby and then the receiver sends the data to a central data collection location for processing. The main purpose of this is to find the optimal placement of the stationary receivers. In the second case, the receivers are mobile, placed inside of a vehicle that is driven from one neighborhood to another. If a vehicle is located within a radius, r, of a meter, the transmission is successful and the usage of gas, water or electricity is recorded. Now, meter readers no longer need to traverse every route in the network, and therefore the TSP is transformed into a CETSP. So, the objective is to create routes that begin and end at the depot and minimize the distance or travel time while passing within a radius of r units of each customer/point. As everyone can understand, mobile receivers have caused a decrease in costs because of the less distance that every vehicle must follow. Therefore, this problem could be viewed as CETSP.

As a project for a graduate course in network optimization, various heuristic for finding near optimal CETSP solutions were developed by six groups of students. In Damon J. Gulczynski, Jeffrey W. Heath, and Carter C. Price [12] paper have been presented the heuristics and have been provided the results for a set of sample cases equal to 100 nodes. Each node/customer is modeled as a point in the plane, where there is a point which represents the depot for the meter reader. The solution to a standard TSP on the customer nodes provides an upper bound for the CETSP.
Essentially someone would think that CETSP is similar to the TSP with time windows, which is a mistake because despite the similarities the problem does not simply change the order of the points visited but change the location of the points themselves.
All the heuristics that have been developed have three distinct steps:

- Given an initial set C of points, the first step is to produce a feasible supernode set S, which is a set of points (containing the depot node) with the property that each point is within r units of at least one point in the set.

- The second step is to find a near optimal TSP tour, T, on the points in S. Since each point is within r units of a supernode, it is guaranteed that in traversing T we will pass within the required radius of each point. Now T is a feasible CETSP tour.

- The third step is an economization routine that reduces the distance traveled in T while maintaining feasibility, thus generating a shorter CETSP tour $\acute{T}$.

The question is how someone could produce the feasible supernode set. Four heuristics were developed for this purpose, Tiling Methods, Steiner Zone, Sweeping Circle and Radial Adjacency. Each of these methods assumes that it is desirable to have as few supernodes as possible.
For the tiling methods the plane is tiled with polygons that can be inscribed in a circle of radius r. Thus, by letting the supernodes be the center of the tiles which contain at least one point, has been created a feasible supernode set. Shifting the centers of the tiles reduces the total number of supernodes and happens when points in two adjacent tiles can be covered by one circle. Specifically, it is taken the maximal and minimal x and y values of the points in the two adjacent tiles $x_{min,}, x_{max}, y_{min}, y_{max}$. The midpoint $\left({}^{x_{min,} + x_{max}}/_{2}, {}^{y_{min}, + y_{max}}/_{2}\right)$ is considered as a potential supernode. If the radial constraints still exist, then the midpoint is accepted and the number of supernodes is reduced by one. This process continues until no mergers are possible.

For the Steiner Zone approach, there is a circle of radius r around every point, representing the point's service region, through which the meter reader must pass. There will be an overlap if the distance between two points is less than 2r. The cardinality of a supernode set can be minimized by choosing points that lie within the intersection of multiple circles. Let D (c_1, r), D (c_2, r), …, D (c_k, r) be discs of radius r centered at points c_1, c_2, …, c_k respectively, then D (c_1, r) ∩ D (c_2, r) ∩ … ∩ D (c_k, r) is the set of points within r units of c_1, c_2, …, c_k. If this intersection is not empty, it is called the Steiner Zone of c_1, c_2, …, c_k and denote it by Z (c_1,

c_2, …, c_k) and k is called the degree of the zone. Every point in the Steiner zone can be a supernode.
The sweeping circle heuristic covers the plane with overlapping circles with centers offset by $d_{min} = min\,(\sqrt{2r}, min(\{d_{i\,j}\}))$, where $d_{i\,j}$ is he Euclidean distance from point i to point j. This method chooses the center of the circle containing the most nodes and adds it to the set of supernodes S in each iteration. All nodes within this circle are covered. These steps are repeated until all nodes in C are covered, at which point S is a feasible supernode set.

For the Radial Adjacency create a matrix containing the distances between each pair of points. Two nodes are said to be adjacent if the distance between them is at most r. An adjacency matrix A is constructed on the points, where entry (i, j) in A is 1 if the points i and j are adjacent, and 0 otherwise. By convention A is 1 along the main diagonal. The degree of a point i is defined as the sum of the entries in row i of A. It is the number of points to which i is adjacent. The supernode set S is created by an iterative method where at each step is considered the point k with the highest degree. The geometric mean of k and all vertices adjacent to k is then computed. If this mean is adjacent to more points than k, then it is designated as a supernode at this step. Otherwise, k is designated as a supernode.

2.3 CONTAINER TERMINAL OPERATIONS

Containers can be used worldwide in several transportation systems. Ship-to-shore links, ship-to-rail links and ship-to-ship links have been developed. In the short-term increased cargo amounts have to be handled at today's container terminals with given limited capacity. High operating costs for ships and container terminals as well as high capitalization of ships, containers and port equipment demand the reduction of unproductive ties at port. The superordinate goal is to reduce the time for the discharging and loading process of a ship. High productivity can be achieved with various types of equipment. Passive vehicles like trucks with trailers or automated guided vehicles are employed in combination with cranes. The strategic decisions focusing on terminal layout, multi-modal interfaces, equipment selection, berthing capacity and information systems are the basis for operational planning as well as the real-time control of crane scheduling ad operation sequencing. Routing and scheduling are the post essential parts of tactical and operational planning.

The complexity of logistics at container terminals demands different concepts, decision rules and optimization algorithms to be compared by simulation before they are implemented into real systems. Most processes at container terminals cannot be foreseen for a long-time span. The time planning horizon for optimization is short and the characteristics of these kind of logistics require online, aka real time, optimization and decision making. For example, the exact time when the containers arrive at the terminal is not known in all cases although data of containers may be pre-advised by electronic data interchange. In general, the container positions within the ship are precisely known in advance and the replanning process allows the calculation of job sequences. But the sequences often have to be changed because of operational disturbances. As vessels are not static and move permanently, because of tide, weather, stability, containers which are next in the sequence cannot be accessed by the cranes spreader. Crane drivers make their own decisions and may alter the pre-calculated operation sequence.

A container terminal can be described as an open system of material flow with two external interfaces. The quayside with loading and unloading of ships and the landside where containers are loaded and unloaded on/off trucks and trains. After arrival at the port, a container vessel is assigned to a berth equipped with cranes to load and unload containers. Unloaded import containers are transported to yard positions near to the place where they are expected to be transshipped next. Export containers arrive by trucks on road or by railway at the terminal. Terminals consist of two components, static stocks, and dynamic transport vehicles. Stocks are defined by their ability to store containers, including yard stacks, ships, trains, and trucks. These different types of stocks differ in capacity and complexity. Transport refers to the transportation of containers in either two or three dimensions.

Different types of cranes are used for the loading and unloading of the ships and trucks (or trains) such as rail mounted gantry cranes (RMGs), rubber-tired gantry cranes (RTG) and overhead bridge cranes (OBC). Although most of the cranes are man-driven, the tendency is for automatic driverless cranes. The transport vehicles can be classified into two different types, the active vehicles, and the passive ones. The active vehicles are able to lift containers by themselves while passive vehicles

must be loaded and unloaded by lifting equipment. The decision on which equipment is used depends on several factors.

Major importance in terminal and in general on logistics applications, have the routing of information and the transportation of data, such as container positioning, the remain capacity of a vehicle, which crane would be used in which condition, or which ship will be serviced first. Most of the time the ports use the First-Come-First-Served option, which does not necessarily minimize the total staying time of ships. While some of the problems at container terminals obviously relate to vehicle routing, others do not, at least at first glance. A problem category is related to ship planning which consists of three partial processes: berth planning, stowage planning and quay crane planning. Since the planning of a ship's stowage does not refer to routing problems, is not going to be analyzed.

Berth scheduling or berth allocation is the process of determining the time and position at which arriving ships will berth. This scheduling process has to become two or three weeks before the ship arrival and in occasion of large overseas ships is known about one year in advance. A typical objective is the minimization of the total sum of shore to yard distances for all containers to be loaded and unloaded, or the minimization of distances traveled by the means of transportation serving the quay cranes. Berth planning can be modeled as Multi -Depot VRP with Time Window (MDVRPTW), where ships represent customers.

Akio Imai, Etsuko Nishimura and Stratos Papadimitriou have published the "The dynamic berth allocation problem for a container port" paper [13], in which is been presented a Lagrangian relaxation-based heuristic algorithm and will be presented in a few words. First of all, a berth allocation problem (BAP) can be approached as static (SBAP) or dynamic (DBAP), with the differences that ships arrive dynamically while work is in progress. In the SBAP all the ships are already in port when the berthing plan is determined, and each berth can service one ship at a time. The objective is the minimization of the sum of waiting time for the availability of the berth assigned to each ship, plus the handling time it spends at the berth. The formulation is as follows:

- Definition of network:
  B let be the set of berths
  V let be the set of ships
  O let be the set of service orders
  C_{ij} let be the handling time spent by ship j at berth i
  A_j let be the arrival time of ship j
  $$S_i = \text{time when berth i becoms idle for the berth allocation planning}$$
  $$(S_i \geq A_j)$$

- Definition of decision variables:
  $$x_{ijk} = = \begin{cases} 1 \text{ if shipj is serviced as the kth ship at berth i} \\ 0 \text{ otherwise} \end{cases}$$

- Definitions of objective function: the minimization of the sum of waiting time for the availability of the berth assigned to each ship, plus the handling time it spends at the berth.

- Definition of model:

  Min $\sum_{i\in B}\sum_{j\in V}\sum_{k\in O}\{(T-K+1)C_{ij}+S_i-A_j\}x_{i\,j\,k}$

  (2.3.1) $\sum_{i\in B}\sum_{k\in O}x_{i\,j\,k}=1,\ \forall\, j\ \in\ V$

  (2.3.2) $\sum_{j\in V}x_{i\,j\,k}\ \leq 1\,,\forall\, i\in B,k\in O$

  (2.3.3) $x_{i\,j\,k}\ \in\{0,1\}\forall\, i\in B,j\in V,k\in O$

  Where the objective function minimizes the sum of waiting and handling time for every ship. The set of constraints 2.3.1 ensures that every ship must be serviced at some berth in any order of service. The set of constraints 2.3.2 ensures that every berth services up to one ship at any time. The set of constraints 2.3.3 ensures that the definition variables are 0 or 1.

For the dynamic problem the formulation is as follows.

- Definition of network:

  B let be the set of berths

  V let be the set of ships

  O let be the set of service orders

  P_k let be the subset of O such that P_k= {p| p < k∈ O

  $C_{i\,j}$ let be the handling time spent by ship j at berth i

  A_j let be arrival time of ship j

  S_i let be the time when berth i becomes idle for the berth allocation planning

  W_i let be the subset of ships with $A_j \geq S_i$ (which is different that the SBAP)

- Definition of decision variables:

  $x_{i\,j\,k}=\begin{cases}1 \text{ if ship j is serviced as the kth ship at berth i}\\ 0 \text{ otherwise}\end{cases}$

  $y_{i\,j\,k}\geq 0$, this variable can have integer values to indicate the time difference between the start of service for ship j and the departure of its immediate predecessor, both of which are integer. To be more specific $y_{i\,j\,k}$ idle time of berth i between the departure of the k-1st ship and the arrival of the kth ship when j is serviced as the kth ship.

- Definition of model:

  Min $\sum_{i\in B}\sum_{j\in V}\sum_{k\in O}\{(T-k+1)C_{I\,j}+S_i-A_j\}x_{I\,j\,k}+\sum_{i\in B}\sum_{j\in W_i}\sum_{k\in O}(T-k+1)y_{I\,j\,k}$

  (2.3.4) $\sum_{I\in B}\sum_{k\in O}x_{I\,j\,k}\ =1\ \forall j\ \in V$

  (2.3.5)$\sum_{j\in V}x_{I\,j\,k}\ \leq 1\ \forall I\ \in B,\forall k\ \in O$

  (2.3.6) $\sum_{l\in B}\sum_{m\in P_k}\left(C_{I\,l\,x_{ilm}}+y_{ilm}\right)+y_{I\,j\,k}-\left(A_j-S_i\right)x_{ijk}\geq 0\ \ \forall i\in B,\forall j\ \in W_i,\forall k\ \in O$

  (2.3.7) $x_{I\,j\,k}\ \in\{0,1,\}\forall\, i\in B,j\in V,k\in O$

  (2.3.8) $y_{i\,j\,k}\ \geq 0,\ \forall\, i\in B,j\in V,k\in O$

The objective function minimizes the sum of waiting and handling times for every ship. As the SPAP modeling, the set of constraints 2.3.4 ensures that every ship must be serviced at some berth in any order of service. The set of constraints 2.3.5 ensures that every berth services up to one ship at any time. The set of constraints 2.3.6 ensure that ships must be serviced after their arrival. The set of

constraints 2.3.7 ensures that the definition variables are 0 or 1 and the set of constraints 2.3.8 ensures that the definition variables $y_{i\,j\,k}$ are equal to or greater than zero.

In the paper is presented the Lagrangian relaxation of the DBAP which is used in the solution method that authors have developed as part of a subgradient procedure. Also, authors have proposed three different processes to find the feasible initial solution, which is the first step and the half of everything: simple, individual and interact. Each procedure works prosthetically to the previous one. Simple process attempts to find the first “unsatisfied ship” in ascending order of service, that is serviced before its arrival. In the individual process no ships are swapped between berths, but if there is an idle time of the berth between two adjacent services of ships, a ship being scheduled for later service is shifted in the idle time as long as it is serviced after its arrival. In the interact process ships are swapped between berths by the condition: the total of its staying time in port and the resulting waiting time of its successors at the new berth is less than that at the original assigned berth. For the computational experiments have generated randomly but systematically problem with 5, 7 and 10 berths, each of which contains data on 25 and 50 ships. The objective function of the relaxed problems is to the total of waiting and handling times spent by incoming ships. For all problems individual process yields the smallest gaps ((feasible solution value- lower bound) * 100/lower bound) which probably identifying better solutions. Although there is no significant difference in overall CPU time.

3 LITERATURE REVIEW

3.1 THE VEHICLE ROUTING PROBLEM (VRP)

The VRP designs the optimal routes used by a fleet of identical vehicles stationed at a central depot to serve a set of customers with known demand one time and then returns to the depot. The basic version of the problem is known as Capacitated VRP (CVRP) in which the objective is to minimize the total cost of the routes under the capacity constraints.

The problem appeared in its primary form in 1959 when G.B. Dantzig and J.H.Ramser [1] published a paper about their solution method for optimization of the minimum routing for the fleet of gasoline delivery vehicles between a terminal (depot) and a specific and large network of service stations (points). In their search the shortest routes between any two points are known and the demand is specific for every point. For a k_i number of vehicles with capacity Q_i (i=1,2,…,n) are required loads q_j to be delivered to every point. Given the distance $d_{i\,j}$ between all points, the main purpose is to minimize the total distance. Obviously, the total load for all routes has to be less or equal to the capacity of each vehicle ($Q_1 \leq \sum_{j=1}^{m} q_j$). It might be noted that if $Q_i \geq \sum_{j=1}^{j=n} q_j$ then the problem becomes the traveling salesman problem, which the authors have used to their solution method, assuming than state $Q_1 = Q_2 = Q_{3=\cdots=} Q_n = 0$. Their method tends to lay more emphasis on filling vehicles to near capacity than on minimizing distance. The calculations could be performed by hand or by automatic computing machine and this is one of the reasons why their method is not used nowadays on practical applications but is very useful for educational purpose.

After that, G. Clarke and J. W.Wright [2] have developed an iterative procedure based on the Dantzi and Ramser solution method, to generate a rapid selection of optimum or near-optimum routes for the same problem, which gave better results. Both methods depend on the variability of the loads of points and also are applicable to small problems such as 30 points with 5 vehicles. This fact is prohibitive for modern applications, for example for an electricity power company which has 200 vehicles and 3000 customers-point for service, so current VRP models aim to incorporate real-life complexities.

An overview of the modern VRP is as follows:

- Definition of network:
  G= (V, E): a graph showing the location and route of the points
  V= {0, 1, …, n}: a location of nodes. The node that represents the depot is 0.
  E: set of edges connecting each node $e_{i\,j}$ = (i, j)
  $d_i \geq 0$: demand of customer i
  $Q \geq 0$: the same maximum vehicle capacity for all vehicles
  $c_{i\,j}$: cost or distance between node i and j

- Definition of decision variables:

$$x_{i\,j} = \begin{cases} 1, & \text{if a vehicle goes from node i to j} \\ 0, & \text{otherwise} \end{cases}$$

- Additional Variable:

$u_i \geq 0$ shows the amount of load that the vehicle is carrying after the visited the point-node I in the route

- Definition of objective function: minimize the sum of travel or distance cost of all vehicles.

- Definition of constraints:

Only one vehicle visits each customer's location.
Depot is the start and return point-node
The total demand of each route does not exceed the capacity of the vehicle

- Formulate according to given constraints and model:

Objective function:

MIN $\sum_{i=0}^{n} \sum_{j=0}^{n} c_{i\,j} x_{i\,j}$

Subject to:

(3.1.1) $\sum_{j \in V\,(j \neq i)} x_{i\,j} = 1\,, \forall\, i \in V$

(3.1.2) $\sum_{i \in V\,(j \neq i)} x_{i\,j} = 1\,, \forall\, j \in V$

(3.1.3) $u_i - u_j + Q x_{i\,j} \leq Q - q_j,\ \forall (i,j) \in A, i \neq 0, j \neq 0$

(3.1.4) $q_i \leq u_i \leq C,\ \forall i \in V$

(3.1.5) $x_{i\,j} \in \{0,1\},\ \forall i,j\ \in V$

(3.1.6) $u_i \geq 0,\ \forall i\ \in V$

FORMULATION 1 VRP

The objective function minimizes the total travel cost. Constraints 3.1.1 ensure that only one vehicle can go from point i to j, constraints 3.1.2 ensure that all vehicles begin their route from depot, constraints 3.1.3 ensure that the number of vehicles coming in and out of a node is the same. Constraints 3.1.4 ensure that the delivery capacity is less than the maximum vehicle capacity, Constraints 3.1.5 are the set of subtour elimination constraints and finally constraints 3.1.6 are the set of decision variables.

A related problem that has received some attention in the literature is the Site-Dependent VRP (SDVRP), in which there is a limited fleet available for the service, no vehicle fixed costs are considered, routing costs are vehicle-independent, and each customer may include restrictions on the vehicle types that may visit it. Some other variants of the problem are the following:

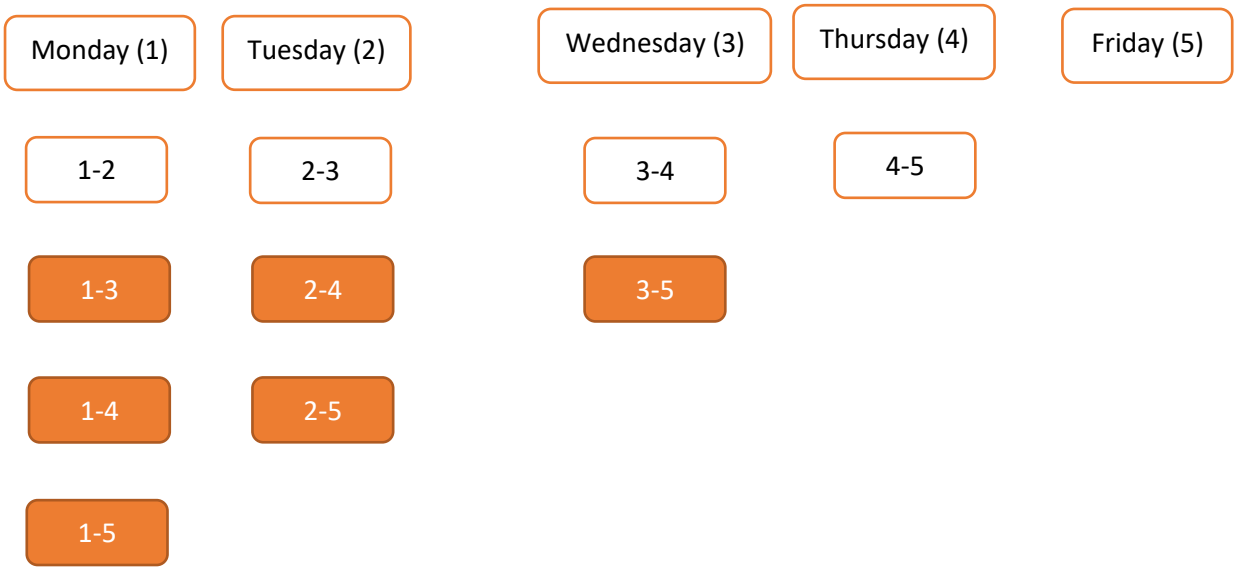
- Period Vehicle Routing Problem (PVRP): the customers have to be serviced in multiple days is a period in order to cover their demand
- Vehicle Routing Problem with Time Windows (VRPTW): the delivery locations have time windows within the visits must be made.
- Vehicle Routing Problem with Pick-Up and Delivery (VRPPD): several goods need to be moved from certain pickup points to other delivery points.
- Vehicle Routing Problem with Profits (VRPP): visit nodes maximize the sum of collected profits under specific circumstances.
- Multi Depot Vehicle Routing Problem (MDVRP): there is more than one depot so the customers could be served from any of the depots using the available vehicles.
- Open Vehicle Routing Problem (OVRP): the vehicles don not return to the depot in the end of the routing and apply to the companies that do not have vehicles at all or the vehicles are not enough to cover the demand. In this situation, most of the times companies hire vehicles and employees in order to realize the distribution of the products.
- Split Delivery Vehicle Routing Problem (SDVRP): a customer is allowed to be serviced by more than one vehicle if this reduces the total cost. It's possible that the customers may be served multiple times. Each vehicle starts and returns to the depot and every time visits a customer, it collects an integer load with the purpose of minimizing the total distance.
- Heterogeneous Fleet Vehicles Routing Problem (HFVRP): the capacity may be different for each vehicle.

In the first phase, the approaches which were presented for the solution of the modern VRP were algorithms based on branch-and-bound, but these methods are still far from being able to solve large instances effectively. Par example, it's possible to solve problems with 50 customers within a reasonable time, but this is still not enough for a practical use. In the second phase, there were mainly classical heuristics, classified into three groups: route construction methods, two-phase methods and route improvement methods. In the third phase, more effective metaheuristic methods were developed which look for a possibly better solution based on an existing solution.

In our days, in the dynamic world of supply chain management, where the VRP and the PVRP belong, the strategic adoption of programming languages has become a defining factor for business aiming to stay in the first raw. Each programming language brings unique capabilities and applications that cater to specific supply chain tasks, from data analysis and automation to statistical modeling and forecasting. Some of these languages are Python, Java, C++, C# and MATLAB with several solvers such as Gurobi, Cplex, Mosek and Xpress.

3.2 THE PERIOD VEHICLE ROUTING PROBLEM (PVRP)

The PVRP (was introduced in the seminal paper by Beltrami and Bodin in 1974) is a generalization of the classic vehicle routing problem in which vehicle routes must be constructed over multiple days (we use “day” as a general unit of time). During each day within the planning period, a fleet of capacitated vehicles travels along routes that begin and end at a single depot. The objective is to find a set of tours for each vehicle that minimizes total travel cost while satisfying vehicle capacity and visit requirements. If a point requires **k** visits during the period, then these visits may only occur in one of a given number of allowable **k-day combinations**. For example, if a point requires two visits during the period (say a 5-day week) the feasible combinations for k =2 is shown in the following diagram. If there is a constraint of being at least one day between the first and last day, then the feasible combinations are in the colored shapes:



According to N. Cristofides and J.E.Beassley paper the data of the problem are as follows:

- Definition of network:

- q_i = demand of customer i (for each delivery)
- c_{ij} = travel time or cost between customer i to j
- Si = the set of allowable combinations of customer i
- N = (i: i=1,2,3, 4…, n) set of customers
- Q_r = capacity for vehicle r
- D_r = total allowable driving time or distance of vehicle r
- R_t = set of available vehicles for day t
- T number of days in the period

- Definition of decision variables

- $a_{kt} = \begin{cases} 1 \ , \text{ if day t is in day} - \text{combination k} \\ 0, \quad \text{otherwise} \end{cases}$

- $u_{ijtr} = \begin{cases} 1 \ , \text{ if vehicle r goes from i to j on day t} \\ 0, \quad \text{otherwise} \end{cases}$

- $x_{ik} = \begin{cases} 1 \ , \text{ if the } \ k^{th} \text{ combination is chosen for customer i} \\ 0, \quad \text{otherwise} \end{cases}$

- $v_{it} = \begin{cases} 1 \ , \text{ if customer i is visited on day t} \\ 0, \quad \text{otherwise} \end{cases}$

  Depot represented by customer 0, so $v_{0t} = 1, \ \forall \, t = 1,2, \dots, T$

- Definition of objective function: minimize the total cost or distance of all vehicles for the period.

- Definition of constraints:

The total demand of the points assigned to day t does not exceed U_d

No more than R_t vehicles are required each day t

The demand load for each vehicle does not exceed Q_r for any route.

The total distance (or time) traveled for any vehicle does not exceed D_r for any route.

Each point is serviced by only one vehicle on each day of period.

The assignment of points requiring service more than once per period satisfies certain day assignment.

Subtour elimination constraints

- Formulate according to given constraints and model:

  Objective function:

  MIN $\sum_{t=1}^{T} \sum_{i=0}^{n} \sum_{j=0}^{n} c_{ij} u_{ijtr}$

  Subject to:

(3.2.1) $\sum_{k \in Si} x_{ik} = 1$ $\forall i\ (i \neq 0)$

(3.2.2) $v_{it} = \sum_{k \in Si} x_{ik}\, a_{kt}$ $\forall t, \forall i\ (i \neq 0)$

(3.2.3) $\sum_{r \in Rt} u_{ijtr} \leq \frac{v_{it}+v_{jt}}{2}$ $\forall i, j, t\ (i \neq j)$

(3.2.4) $\sum_{i=0}^{n} u_{iptr} = \sum_{j=0}^{n} u_{pjtr}$ $\forall p, t, r \in Rt$ (p-day period)

(3.2.5) $\sum_{r \in Rt} \sum_{i=0}^{n} u_{ijtr} = \begin{cases} v_{jt}, & \forall j, t\ (j \neq 0) \\ |Rt|, & \forall t\ (j = 0) \end{cases}$

(3.2.6) $\sum_{i,j} u_{ijtr} \leq |W| - 1$ $\forall W \subseteq N$ and $2 < |W| < n-1$; $t, r \in Rt$

(3.2.7) $\sum_{j=1} u_{0jtr} \leq 1$ $\forall t, r \in Rt$

(3.2.8) $\sum_{i=1} q_i \left(\sum_{j=0} u_{ijtr}\right) \leq Q_r$ $\forall t, r \in Rt$

(3.2.9) $\sum_{i,j} c_{ij}\, u_{ijtr} \leq D_r$ $\forall t, r \in Rt$

(3.2.10) $x_{ik} \in \{0, 1\}$ $\forall i, k \in St$

(3.2.11) $u_{ijtr} \in \{0, 1\}$ $\forall i, j, t, r \in Rt$

FORMULATION 2 PVRP

The objective function minimizes the total travel cost or the total travel time or the total travelled distance. Constraints 3.2.1 ensure that only one delivery combination is chosen for each point, constraints 3.2.2 ensure that a point is only visited on a particular day if the delivery combination chosen has a delivery on that day, constraints 3.2.3 ensure that no vehicle can go between two points on a particular day unless they are both scheduled for delivery on that day, constraints 3.2.4 ensure that if a vehicle visits a point, it also leaves that point. In constraints 3.2.5 the upper half represents that each point is visited on the days that it is scheduled for delivery and the lower half that all available vehicles in a day will return in depot. Constraints 3.2.6 are the set of subtour elimination constraints.

Constraints 3.2.7 ensure that a vehicle can only be used at most once, constraints 3.2.8 and 3.2.9 ensure that the vehicle capacity and the vehicle time will not be violated, respectively. Finally, constraints 3.2.10 and 3.2.11 define the set of decision variables.

3.3 RELAXATIONS (PMP, PTSP) AND VARIANTS (MDPVRP, PVRPTW, PVRPSC) OF PVRP

The program for the PVRP is a complex '0-1' program involving an enormous number of variables. It's not possible to be solved except for trivially small cases, so one way of solving this would be to relax the VRP for each day of the period. In accordance with Christofides and Beasley there are two relaxations where the subproblem for each day becomes a) a median problem (PMP) and b) a traveling salesman problem (PTSP).

a) PMP: The expected length of vehicle routes in a VRP is related to the sum of the radial distances of the points from a center. We would therefore expect that minimizing the sum of the radial distances of the customers from a center chosen for each day of the period would also tend to minimize the underlying PVRP. The formulation is following:

d_{ij} = radial distance of point i from center j

J_t = set of centers on day t

$$y_{jt} = \begin{cases} 1, & \text{if j is a center on day t} \\ 0, & \text{otherwise} \end{cases}$$

$$z_{ijt} = \begin{cases} 1, & \text{if i is assigned to center j on day t} \\ 0, & \text{otherwise} \end{cases}$$

$$x_{ik} = \begin{cases} 1, & \text{if the } k^{th} \text{ combination is shosen for customer i} \\ 0, & \text{otherwise} \end{cases}$$

Objective function:

Min $\sum_t \sum_i \sum_j d_{ij} z_{ijt}$

Subject to:

(3.3.1) $\sum_{k \in Si} x_{ik} = 1$ $\forall$ i

(3.3.2) $\sum_j z_{ijt} = \sum_k x_{ik} a_{kt}$ $\forall$ i, t

(3.3.3) $\sum_i q_i z_{ijt} \le (\sum_r Q_r) y_{it}$ $\forall$ t, j $\in J_t$

(3.3.4) $\sum_{j \in Jt} y_{jt} = 1$ $\forall$ t

(3.3.5) $x_{ik} \in (0,1)$ $\forall$ i, k $\in$ St

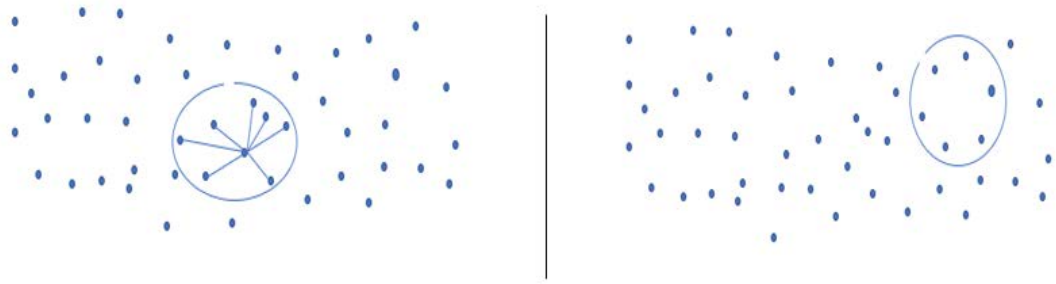
(3.3.6) $z_{ijt} \in (0,1)$ $\forall$ i, t, j $\in$ Jt

FOMULATION 3 PMP

The objective function minimizes the total distance. Constraints 2.3.1 ensure that only one combination is chosen for each point. Constraints 2.3.2 ensure that a point is only allocated to a center on a particular day if it is visited on that day. Constraints 2.3.3 ensure that the total demand the points visited on a particular day does not exceed the total available capacity for that day and

that a point cannot be allocated to a point that has not been picked as a center. Constraints 2.3.4 ensure that only one center is chosen for each day. Constraints 2.3.5 and 2.3.6 define the set of decision variables.

Someone could question what happens when there is a cycle of points, as an example is the following figures.



In both cases is easy to choose a point center. In the left case it's obvious that the point center is the point near the center of the cycle. In the right case the point center could be every point of the set, because the sum of distances between the points would be the same regardless of point center.

b) PTSP: The travelling salesman problem is to find the shortest route to visits each point once before returning to the starting point. Someone would expect that the length of the traveling salesman tour would be related to the length of the vehicle routing solution. As a follow-up to it, someone would expect that minimizing the sum of the lengths of the traveling salesman tours for each day of the period would also tend to minimize the underlying PVRP. The program for this period TSP according to Miller-Tucker- Zemlin [6] is following:

$$x_{ij} = \begin{cases} 1 & \text{if the route includes a direct link between points i and j} \\ 0 & \text{otherwise} \end{cases}$$

Objective function:

Min $\sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}\, x_{ij}$

Subject to:

(3.3.7) $\sum_i x_{ij} = 1 \qquad \forall j$

(3.3.8) $\sum_j x_{ij} = 1 \qquad \forall i$

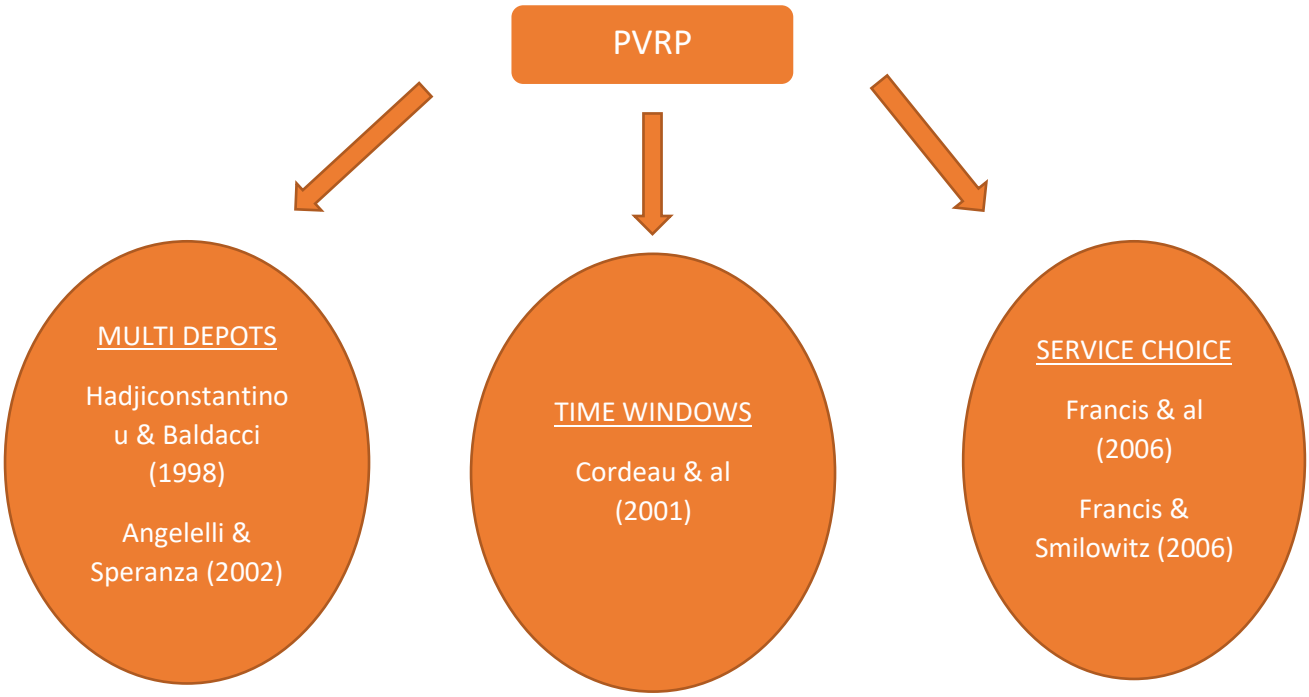
(3.3.9) $u_i + x_{ij} \leq u_j + (n-1) * (1 - x_{ij}) \qquad 2 < j \leq n$

(3.3.10) $u_i \in R^+$

(3.3.11) $x_{ij} \in \{0,1\}$

FORMULATION 4 PTSP

The objective function minimizes the total cost. Constraints 3.3.7 are named Go-to constraints and require that each point is arrived at from exactly one other point. Constraints 3.3.8 are named Come-from constraints and require that from each point there is a departure to exactly one next point. Constraints 3.3.9 are subtour elimination constraints and constraints 3.3.10 and 3.3.11 are the sets of decision variables.



According to the figure above there are also 3 very important variants of the initial problem, c) the Multiple Depot PVRP (MDPVRP), d) the PVRP with Time Window (PVRPTW) and e) the PVRP with Service Choice (PVRPSC). All those variants have analogs in single-visit VRP models.

c) MDPVRP: in accordance with the paper of Hadjiconstantinou and Baldacci [7] the first problem is to find the most efficient boundaries of the geographical area served by each depot to achieve a specified level of service. The second problem is to determine, over a given period, optimal visiting patterns for the fleet of vehicles, that is, to plan the scheduling of the vehicle visits to all points within each depot service area in the best possible way. The third problem is to perform efficient route planning for the vehicles within the optimal depot territories subject to a variety of constraints.

d) PVRPTW: in this problem there are time window constraints, in which points can be visited only at specific time interval during the planning period. Cordeau, Laporte and Mercier [8], in their paper proposed a Tabu algorithm to solve the problem, which is a local search metaheuristic that explores the solution space by moving at each iteration from a solution s to the best solution in its neighborhood N(s). The problem is defined on a complete graph G= (V, A). The set V = $\{u_0, u_1, \ldots, u_n\}$ is the vertex set where u_0 is the depot and the others are the points which must be visited and A = $\{(u_i, u_j): u_i, u_j \in V, i \neq j\}$ is the arc set. With each vertex $u_i \in V$ are associated a nonnegative load q_i, (with $q_0 = 0$), a nonnegative service duration d_i, (with $d_0 = 0$), and a time window $[e_i, l_i]$, where e_i is the lower bound (nonnegative integer) and l_i is the upper bound (nonnegative integer). Each arc (u_i, u_j) has an associated nonnegative cost or travel time c_{ij}. A planning horizon of t days is considered, and each point i specifies a service frequency f as well as a set of R_i allowable combinations of visit days. The fleet consists of m vehicles, so the problem focus on designing m routes on G graph under constraints such as: every route starts and ends at the depot, every point

belongs to exactly one route, the total load and duration of route k do not exceed Q_k and D_k respectively, the service at point i begins in the interval $[e_i, l_i]$, every vehicle leaves the depot and returns to the depot in the interval $[e_0, l_0]$.

e) PVRPSC: in this problem the visit frequency of points is a decision of the model. Francis and Smilowitz [9] have worked on this project and present a continuous approximation model to facilitate strategic planning of period distribution systems with continuous function to describe service allocation and vehicle routes due to overcome difficulties such as exponential growth of variables and constraints and small-scale problems that can be solved by methods developed by Francies et al (for example sizes such as fifty nodes, three service choice and five days). In this section, we are going to present the description of the model that has been developed by Francis et al and in the next section of solution methods we are going to present the continuous approximation that Francis and Smilowitz have developed.

The givens of the problem are the set of nodes with known demand and minimum visit frequency requiring service over the planning period, the vehicle fleet, the network travel times and the set of service schedules with headways and benefits. In more detail let S be the set of schedules and D be the set of days in the period. Let the parameter $a_{sd} = \begin{cases} 1, & \text{if } d \in D \text{ is in schedule } s \in S \\ 0, & \text{otherwise} \end{cases}$, and γ^s be the visit frequency associated with each schedule $s \in S$, which is measured by the number of days in the schedule as $\gamma^s = \sum_{d \in D} a_{sd}$. Moreover, for every schedule there are 2 givens, the $H^s = {}^1/_{\gamma^s}$ which is the headway between visits and the benefit a^s related to the monetary benefit of more frequent service that is assumed to be stationary over the period. Let N be the set of nodes, in which i=0 is the depot. Let A={(i,j): i,j ∈ N} be the set of connecting arcs of the nodes, let W_i be the daily demand for every node and F_i be the minimum service frequency measured in days per period. Let ${w_i}^s$ be the demand accumulated between visits (is a schedule and daily demand function for every node). Let ${\tau_i}^s$ be the stopping time at a node which is a function of the frequency of the schedule since more items accumulate with less frequent service and, therefore, require more time to load/ unload. Let $c_{i\,j}$ be the travel cost for going to node j from node i, K be the set of vehicles and C the capacity of each vehicle. There are two sets of decision variables, and the formulation is following:

$${y_{i\,k}}^s = \begin{cases} 1, & \text{if node i is visited by vehicle k on schedule s} \\ 0, & \text{otherwise} \end{cases}$$

$${x_{i\,j\,k}}^d = \begin{cases} 1, & \text{if vehicle k traverse arc (i, j) on day d} \\ 0, & \text{otherwise} \end{cases}$$

Objective function:

$$\text{Min } Z = \sum_{k \in K}\left[\sum_{d \in D} \sum_{(i,j) \in A} c_{i\,j}\, {x_{i\,j\,k}}^d + \sum_{s \in S} \sum_{i \in N} \left(\gamma^s {\tau_i}^s - W_i a^s \right) {y_{i\,k}}^s \right]$$

Subject to:

$$(3.1.1)\ \sum_{s\in S}\sum_{k\in K}\gamma^{s}y_{i\,k}{}^{s}\geq F_{i}, \qquad \forall\, i\in N$$
$$(3.1.2)\ \sum_{s\in S}\sum_{k\in K}y_{i\,k}{}^{s}\leq 1, \qquad \forall\, i\in N$$
$$(3.1.3)\ \sum_{s\in S}\sum_{i\in N}w_{i}{}^{s}\,a_{sd}y_{i\,k}{}^{s}\leq C, \qquad \forall\, k\in K, d\in D$$
$$(3.1.4)\ \sum_{j\in N0}x_{i\,j\,k}{}^{d}=\sum_{s\in S}a_{s\,d}y_{i\,k}{}^{s}, \qquad \forall\, i\in N, k\in K, d\in D$$
$$(3.1.5)\ \sum_{j\in N0}x_{i\,j\,k}{}^{d}=\sum_{j\in N0}x_{j\,i\,k}{}^{d}, \qquad \forall\, i\in N0, k\in K, d\in D$$
$$(3.1.6)\ \sum_{i,j\in Q}x_{i\,j\,k}{}^{d}\leq |Q|-1, \qquad \forall QCN, k\in K, d\in D$$
$$(3.1.7)\ y_{i\,k}{}^{s}\in\{0,1\}, \qquad \forall\, i\in N, k\in K, s\in S$$
$$(3.1.8)\ x_{i\,j\,k}{}^{d}\in(0,1), \qquad \forall(i,j)\in A, k\in K, d\in D$$

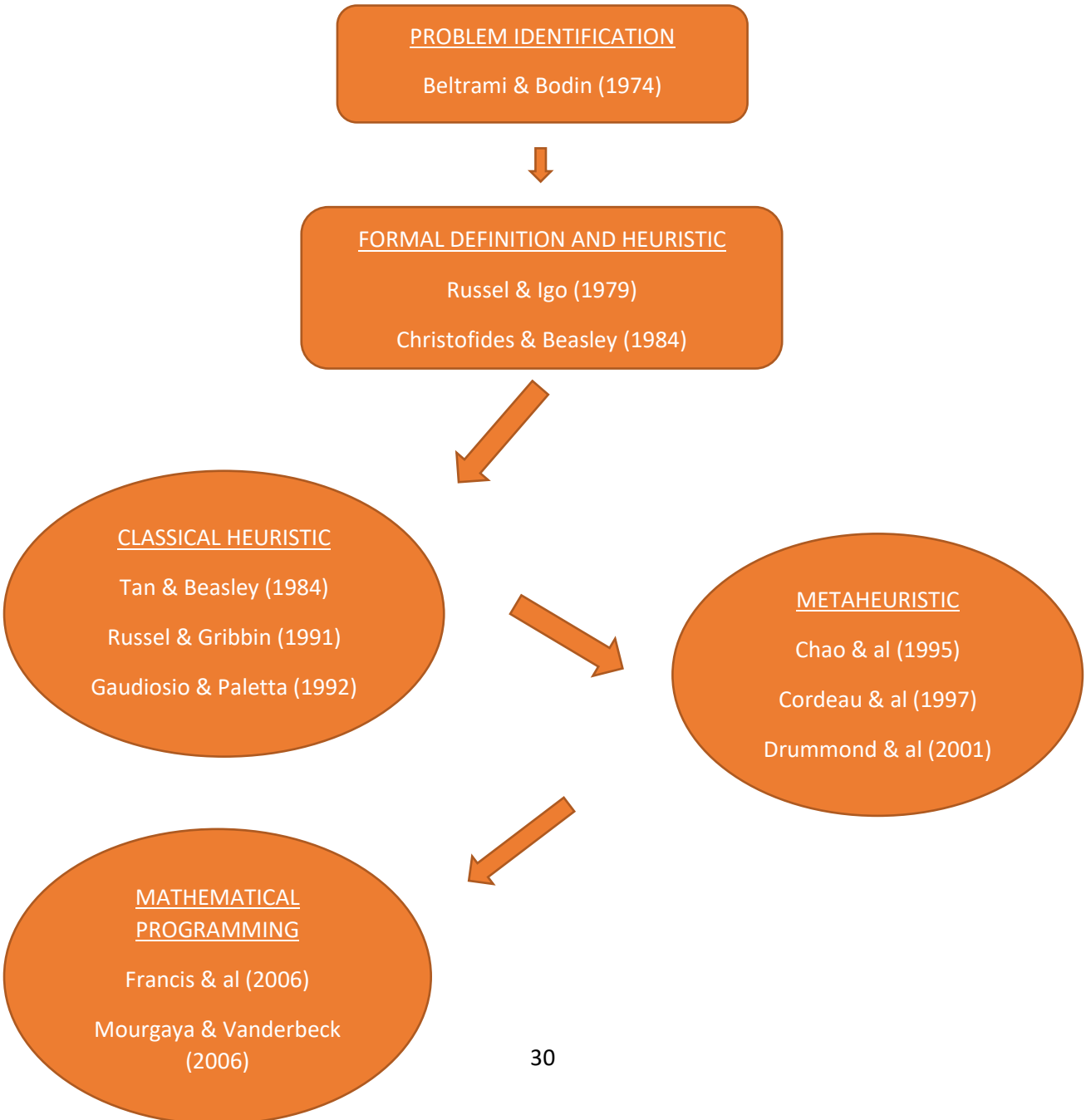
FORMULATION 5 PVRPSC

The objective function balances arc travel times, stopping times and demand-weighted service benefit. Constraints 3.1.1 enforce the minimum visit frequency for each node. Constraints 3.1.2 ensure that one schedule and one vehicle are chosen for each demand node. Constraints 3.1.3 represent vehicle capacity constraints. Constraints 3.1.4 link the x and y variables for the demand node. Constraints 3.1.5 ensure flow conservation at each node. Constraints 3.1.6 are the subtour elimination constraints and ensure that all tours contain a visit to the depot. Constraints 3.1.7 and 3.1.8 define the binary variables for allocation and routing, respectively.

4 SOLUTION METHODS

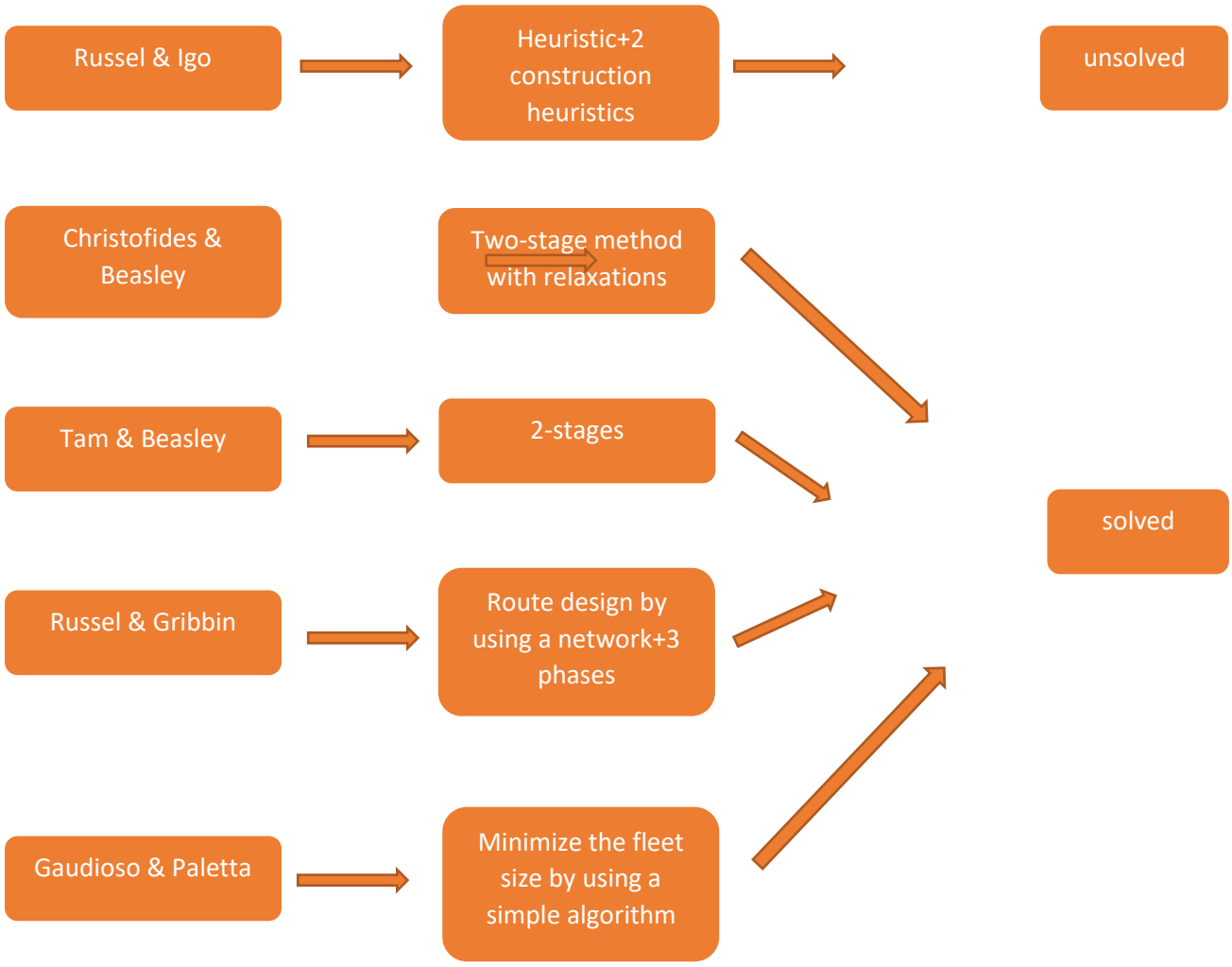
4.1 SOLUTION METHODS FOR PVRP

In the diagram below is shown the evolution of the solution approaches and the reacershers of them for dolving the PVRP. Classical Heuristics tend to solve the assignment and routing decisions sequentially and are not effective for problems with a big number of points. After that literature has focused on metaheuristic methods that can escape the trap of local optimality that plagues conventional heuristics. These methods use an initial feasible solution to the PVRP and then improve this until find the optimal solution to the problem. Nevertheless, the complexity of the problem requires a combination of approaches which is called mathematical programming and is based on relaxations of the initial problem. Nowadays, researchers use programming languages such as c^{++} and python.



4.2 CLASSICAL HEURISTICS FOR PVRP

Heuristic is the first step that someone should follow to solve the PVRP and the variants of it. In the figure below is shown the researchers and the roads that they followed to solve the problem or to improve the formulation of it.



Russel and Igo consider three heuristics for the problem. The first method is a cluster algorithm, where clusters are formed around customers that are scheduled for delivery on each day of the period. Once the delivery combinations have been decided, a vehicle routing algorithm is applied to each day of the period separately. The second algorithm takes the routes and attempts to improve them by a sequence of link exchanges. The exchange algorithm was the MTOUR heuristic of Russell, which is itself a generalization of the X-optimal traveling salesman heuristic of Lin and Kernighan. The third algorithm is a modification of the Clarke and Wright savings algorithm. Points are initially allocated to days of the week in accordance with the delivery spacing constraints and routes are then formed using the savings algorithm, with the modification that a link is not made between two points if to do so would violate the delivery spacing constraints.

In accordance with Christofides and Beasley paper, they proposed relaxations which they did not solve optimally. They used a two-stage heuristic method: they allocated nodes to days and after that they attempted to exchanges with the aim of minimizing the cost. The first stage is to find an initial allocation of routes. Specifically, step A.1 (form an ordering of the customers) is to form a list of points arranged in descending order of 'importance', placing all points with fixed delivery combinations at the top. For the remaining points was used the heuristic of demand order in which the points with large demands are first and that helps avoid serious feasibility problems. Step A.2 (assign delivery combination) is to assign delivery combinations by following a mini algorithm. Assign points into a list and for each one (a) evaluate the increase in total cost for the period for each combination when the resulting deliveries are added to those from previous assignments, (b) choose the feasible combination that gives the least overall cost increase and (c) if no combination satisfies the constraints, assign these that gives the least overall cost increase. Step A.3 (local optimization) is to apply a local optimization procedure to each day separately to improve the cost for that day. For the second stage is necessary to choose a small subset of points and enumerate the delivery combination possibilities for this set, seeking an improved overall solution. Step B.1 is to define the family $\mathfrak{U}$ of subsets of points. In step B.2 (seek improved solution) for every $U \in \mathfrak{U}$ perform the following:

(a) Remove points in U from problem. For $i \in U$
- k^*_i: represent the current delivery combination of i
- d*: be the current total delivery cost

Temporarily delete the current assigned delivery combinations $k^*_i \ \forall i \in U$. This is equivalent to removing the customers $i \in U$ from the days on which they were scheduled for delivery so that the quantity delivered on any day and the delivery cost for that day will decrease.

(b) Enumerate delivery combination possibilities of points in U. Generate all possible delivery combination sets for the set of points U and for each set of delivery combinations follow the next step.

(c) Evaluate delivery combination possibilities. Let the delivery combination set being considered be represented by $K = (k_i \mid i \in U)$. If the total delivery cost with this combination set K is $d < d^*$ and the solution for each day (appearing in K) is feasible, then we have an improved solution. We can then change the delivery combinations for the set U of customers to K, and update d*.

In step B.3 (termination rule) if there is not improvement by step B.2 stop. The current point combinations form the final assignment. If some improvement has occurred, then (a) perform the local optimization procedure as at step A.3 and (b) go to step B.1 to redefine the family of points sets $\mathfrak{A}$ unless an upper limit on the number of iterations has been reached, in which case stop.

4.2.A. ADAPTATION FOR THE PMP

The first step to adapt the general heuristic method to a period median problem is to generate T centers (one for each day of the period). Each point could be assigned the delivery combination that gave the least overall cost of allocation to give an initial solution. This solution is improved using point pair interchanges. In more detail the method is as follows:

1. Define T centers:
   i. J set of centers and initially J=∅. Perform J= (J ∪ i^*), T times where i^* corresponds to the value of I which produces the maximum to $i=1,\dots,n \max_{i \notin J} [d_{0\,i}\ (\prod_{j \in J} d_{ij})]$. Thus, the fist center that is chosen is the furthest from the depot, then the point furthest from the depot and the first center until T spaced centers have been founded. A day has to be associated with each center as follows:
   ii. For each point i with frequency of delivery f_i, define J_i as the f_i nearest centers to i ($J_i C\ j\ \forall\ i$).
   iii. $m_{j\,t} = \sum_{j \in J_i} [(\sum_{k \in S_i} a_{k\,t})/|S_i|]$. Form the matrix $m_{j\,t}$ j ∈ J, t=1,…, T. For a given center j, form $m_{j\,t}$ from all points i for whom j is one of their fi nearest centers. The larger $m_{j\,t}$ is, the more attractive it is to associate day t with the center j.
   iv. Solve the assignment (maximization) problem for the "attractiveness" matrix. Note that this is a small **(T** X **T)** linear assignment problem.

2. Assign Delivery Combinations: Let j (t) be the center of day t as decided in step.1 and then the cost of combination k for point i is the following and the least cost combination is chosen for each customer.

$$\sum_{t=1}^{T} d_{i\,j\,(t)} a_{k\,t} \quad , \forall\ i,k\ \in\ S_i$$

3. Relocation of Centers: Let a center be associated with each day t, and a set of points which must be visited on day t. The center for each day can be relocated to coincide with the optimal one-median of the points to be visited on that day. Although no combination is changed for any point.
4. Interchanges: In the computational results, have been considered all possible pairs of points for point combination interchanges. There is $\mathfrak{A}$ = ((i, j), i=1…,n and j =1,…,n with i<j) .

4.2.B. ADAPTATION FOR THE PTSP

It is necessary to get as a starting point the delivery combinations chosen in section of the adaptation for the PMP. For each day of the period, we then formed an initial tour of the points using a least cost insertion heuristic.

Perform points interchanges to improve the initial solution. The rule for selecting the family $\mathfrak{U}$ is as follows: each set of $\mathfrak{U}$ is the union of 2 other sets M1, is a set of points appearing consecutively on the tour for some day t, and M2 is a set of points near the set M1.

i. Choose any arbitrary point i on a tour for some day t, and L points that follow i on the tour. Point i and the set of L constitute the M1, which is a set of points expected to be physically near to each other. Now choose a second set M2 that lie on different tours from the tour the set M1 are on but are physically near to M1.

ii. Let α and β be the 2 points that precede and follow M1 on the tour for day t and let their current delivery combinations be k^*_α and k^*_β . Generate a list of all points $j \notin M1$ ordered by:

- Increasing order of the number of delivery days for j that coincide with delivery days for α or β

$$\sum_{t=1}^{T} a_{k^*_j} \min\{1,\ a_{k^*_\alpha t} + a_{k^*_\beta t}\}$$

- Ties at above broken by ordering in increasing distance from α and β $c_{aj} + c_{j\beta}$

iii. The first L^1 points of this ordered list constitute the set M2 with M1 ∪ M2 being one set of points in the family of point sets. The complete family is generated by considering all days t and all points i on the tour for day t.

4.2.C. ADAPTATION FOR THE MDPVRP

In the paper of Hadjiconstantinou and Baldacci [7] is presented the solution method for this problem as a generalization of the classical heuristic of the VRP, which involves a given set of points, with specific service requirements, that must be visited by vehicles started by a single depot. The generalization includes several depots instead of only one and the panning horizon is not a single day but several days. First, define boundaries for each depot service area. Second, solve a PVRP for each depot. Third, solve a classical VRP for each depot and for each day of the planning period, Fourth, solve a classical Traveling Salesman Problem (TSP) for each route. Proportionally the solving of MDPVRP is as hard as TSP's. In this thesis the TSP has been described in the paragraph about the variants. The steps that authors proposed for solve the variant are as follows:

1. Form an ordering of the points
2. Assign each point to its nearest depot and evaluate visit combinations:

a) Arrange the nodes according to some chosen order for example the decreasing visit frequency.

b) Using this ordered list, use least-cost insertion to add points to routes such that their visit frequency fi is satisfied.

c) When all nodes have been assigned to feasible route combinations, solve VRPs for each day of the planning period using Tabu method. This algorithm has been used by a lot of authors and is a metaheuristic method that improves the solution by looking for a better one in a local (neighborhood) search. This search stops when a criterion is violated.

3. Attempt to improve the overall solution by interchanging point routes and/or assignments.

4.2.D. ADAPTATION FOR THE PVRPSC

In this section we are going to present the model that Francis and Smilowitz have followed for the period vehicle routing problem with service choice, which in short is a variant of the classic PVRP where the visit frequency of points is a decision of the model. They chose to use continuous functions (a function whose graph is continuous without any breaks or jumps) to describe variables and parameters, which means that we do not have specific data for the network's nodes or demand, but volumes. So, we do not use a set of nodes but e region of node in a service region R.

As the model of Francis et al, which has been presented in the section above,, we keep the set of days (D) in the period, the capacity C of vehicles (same for every vehicle) , the parameter $\mathrm{a_{sd}} = \begin{cases} 1, & \text{if d} \in \text{D is in schedule s} \in \text{S} \\ 0, & \text{otherwise} \end{cases}$, the visit frequency, $\gamma^{\mathrm{s}} = \sum_{\mathrm{d}\in\mathrm{D}} \mathrm{a_{sd}}$. Moreover, for every schedule there are 2 givens, the $\mathrm{H^s} = {}^{1}\!/_{\gamma^{\mathrm{s}}}$ which is the headway between visits and the benefit a^s related to the monetary benefit of more frequent service that is assumed to be stationary over the period

The new parameters and givens are the bellow:

- A: is a subregion of service region R
- The spatial density of nodes with minimum service schedule m ∈ S about a point x: $\delta\ (x)^m$
- Number of nodes on A: for a subregion A of service region R, the number of nodes would be N(A)= $\sum_{i\in S}\left(\int_{x\in A}\delta^m(x)\,dx\right)$
- The demand density rate about a point x of nodes measured in items per unit time area: $\lambda\ (x)^m$
- Fraction of nodes about a point x with minimum schedule m being served by service schedule s: $f\ (x)^{ms}$
- Spatial density of nodes about a point x to be visited on day d (measured in nodes per unit aera): $\Delta(x)^d$
- Demand density on day d (measured in demand per unit area): $\Lambda(x)^d$
- Number of stops on a route on day d of period: $n(x)^d$
- Shipment size collected at a node on day d of period: $u\ (x)^d$
- Distance from the depot to point x: r(x)
- Average distance between nodes: ${}^{\hat{k}}\!/_{\sqrt{\Delta(x)^d}}$, where $\hat{k}$ is a metric-depended constant

- Average cost per distance $\bar{c}$ is the average of c_{ij} divided by the distance between nodes I kai j over all (i,j) ∈ A
- Stopping cost:$\tau_i{}^s = \dot{\tau} + \ddot{\tau}\, u(x)^d$, where $\dot{\tau}$ is a fixed stopping cost and $\ddot{\tau}$ is a variable cost that increases with the demand accumulated at node i.
- Routing cost per item on day d of period:

$$z(x)^d = {2r(x)\bar{c}}\Big/{n(x)^d u(x)^d} + {\bar{c}\hat{k}\sqrt{\Delta(x)^d} + \tau_i{}^s}\Big/{u(x)^d}$$

Formulation:

$$Z_R = \int \left[\sum_{d \in D} \Lambda(x)^d z(x)^d - \sum_{s \in S} a^s \sum_{m \in S} \lambda(x)^m \, f(x)^{ms} \right] dx$$

Subject to:

(4.2.1) $n(x)^d u(x)^d \leq C, \forall\, d \epsilon D, x \epsilon R$

(4.2.2) $u(x)^d = {\Lambda(x)^d}\Big/{\Delta(x)^d}, \forall d \in D, x \in R$

(4.2.3) $\Delta(x)^d = \sum_{s \in S} a_{sd} \sum_{m \in S} \delta(x)^m f(x)^{ms}, \forall d \in D, x \in R$

(4.2.4) $\Lambda(x)^d = \sum_{s \in S} a_{sd} H^s \sum_{m \in S} \lambda(x)^m f(x)^{ms}, \forall d \in D, x \in R$

(4.2.5) $\sum_{s \in S} f(x)^{ms} = 1, \ \forall m \in S, x \in R$

(4.2.6) $0 \leq f(x)^{ms} \leq 1, \forall m \in S, s \in R : \gamma^m \leq \gamma^s$

(4.2.7) $f(x)^{ms} = 0, \ \forall m \in S, s \in R : \gamma^m > \gamma^s$

(4.2.8) $n^d \geq 0, \ \forall d \in D, x \in R$

FORMULATION 6 PVRPSC

The main purpose of he objective function is to minimize the total cost over all schedules. Constraints 4.2.1 ensure that the shipment size does not exceed the capacity of vehicles. Constraints 4.2.2 ensure that the shipment size is equal to the demand on the point x. Constraints 4.2.3 define the effective density od nodes visited per unit area on day d. Constraints 4.2.4 define the effective demand density collected per unit area on day d. Constraints 4.2.5 ensure than every node will be serviced by a schedule. Constraints 4.2.6 and Constraints 4.2.7. ensure that every node will be serviced with frequency no less than the minimum specified. Constraints 4.2.8 ensure that the stops on day d cannot be negative.

To start solving the above formulation, the authors started by decomposing the geographic area of the problem into subregions, which should be small enough such that the spatial density of nodes, $\delta\ (x)^m$, and the demand density rate, $\lambda\ (x)^m$, are almost constant over the subregion A of R and large enough to contain at least one route. This decomposing allows the objective function and constraints for each subregion be independent of the other subregion. They define the cost in every subregion as $\zeta_A = \sum_{d\in D} \Lambda^d z^d - \sum_{s\in S} \alpha^s \sum_{m\in S} \lambda^m f^{ms}$ and sum over all subregions costs to obtain the total one. This definition simplifies the formulation above such as:

$$\sum_{d\in D} \Lambda^d z^d = \sum_{d\in D} \Lambda^d \left[\frac{2r\bar{c}}{n^d u^d} + \frac{\bar{c}\hat{k}\ (\sqrt{\Delta^d} + \dot{t} + \ddot{t}u^d}{u^d}\right]$$

Since there are no limits on the number of vehicles or the distance traveled by a vehicle, an optimal solution will choose to transport full vehicle capacity C= $n^d u^d$, which means that the constraints 4.2.1 are already binding. Moreover, since the sum over all days of Λ^d is equal to the total demand over the planning period which must be served, they made the constant

$\beta_0 = ({}^{2r\bar{c}}/_{C} + \ddot{t})\ \sum_{m\in S} \lambda^m$, so, the function above becomes

$\sum_{d\in D} \Lambda^d z^d = \beta_0 + \dot{t} \sum_{d\in D} \Delta^d + c\hat{k} \sum_{d\in D} \sqrt{\Delta^d}$

For the cost of every subregion, we have:

$\zeta_A = \beta_0 + \dot{t} \sum_{d\in D} \Delta^d + c\hat{k} \sum_{d\in D} \sqrt{\Delta^d}$ - $\sum_{s\in S} \alpha^s \sum_{m\in S} \lambda^m f^{ms}$ =>

$$\beta_0 + \dot{t} \sum_{d\in D} \sum_{s\in S} a_{sd} \sum_{m\in S} \delta^m f^{ms} + c\hat{k} \sum_{d\in D} \sqrt{\sum_{m\in S} \delta^m f^{ms}} - \sum_{s\in S} \alpha^s \sum_{m\in S} \lambda^m f^{ms} =>$$

$\beta_0 + \dot{t} \sum_{s\in S} \gamma^s \sum_{m\in S} \delta^m f^{ms} + c\hat{k} \sum_{d\in D} \sqrt{\sum_{m\in S} \delta^m f^{ms}} =>$

$\beta_0 + \dot{t} \sum_{s\in S} \gamma^s \sum_{m\in S} \delta^m f^{ms} + c\hat{k} \sum_{d\in D} \sqrt{\sum_{m\in S} \alpha^s \sum_{m\in S} \lambda^m f^{ms}} - \sum_{s\in S} \alpha^s \sum_{m\in S} \lambda^m f^{ms}$ =>

$\beta_0 + \sum_{m\in S} \sum_{s\in S} (\dot{t}\delta^m\gamma^s - \alpha^s\lambda^m) f^{ms} + \sum_{d\in D} \sqrt{\sum_{s\in S} (c\hat{k})^2 a_{sd} \sum_{m\in S} \delta^m f^{ms}}$ =>

$\beta_0 + \sum_{m\in S} \sum_{s\in S} (\dot{t}\delta^m\gamma^s - \alpha^s\lambda^m) f^{ms} + \sum_{d\in D} \sqrt{\sum_{m\in S} \sum_{s\in S} (c\hat{k})^2 a_{sd}\delta^m f^{ms}}$ =>

$\beta_0 + \sum_{m\in S} \sum_{s\in S} \beta_1 f^{ms} + \sum_{d\in D} \sqrt{\sum_{m\in S} \sum_{s\in S} \beta_2 f^{ms}}$

So, the optimization problem to be solved for each subregion is:

$$\text{Min } \zeta_A = \beta_0 + \sum_{m\in S} \sum_{s\in S} \beta_1 f^{ms} + \sum_{d\in D} \sqrt{\sum_{m\in S} \sum_{s\in S} \beta_2 f^{ms}}$$

Subject to:

(4.2.9) $\sum_{s\in S} f^{ms} = 1, \forall m \in S$

(4.2.10) $f^{ms} = 0\ , \forall m.\ s \in S: \gamma^m > \gamma^s$

(4.2.11) $0 \leq f(x)^{ms} \leq 1, \forall m \in S, s \in R:\ \gamma^m \leq \gamma^s$

With this approach the authors turned the original problem into a non-linear one which could be solved by any available non-linear solver, such as AMPL with the KNITRO solver. For those who don't know, AMPL is a programming language for optimize linear, non-linear and integer models with a variety of solvers such as CPLEX, KNITRO and GUROBI.

They choose to solve the 100 b PVRP data set from Christofides and Beasly [3] (with problem details as follows: number of customers:100, length of period (days): 5, vehicles each day: 5, if demand ≤10 then we have one delivery in period, if 11≤demand≤25 then we have two delivery in period and if demand ≥ 26 we have delivery every day) to compare the best known discrete solution with the solution obtained with their algorithm without service choice, so they can show the benefit of service choice. They split the region into three (3) subregions such that the node densities and demand rates are almost uniform within each region. Every subregion is large enough to have at least one route. To account for the travel distance to and from the depot, the depot is added in the calculation of node density for daily service. They consider three service schedules, m=1,2,3, where 1 represents one-day service per week, 2 represents two-day service per week and 3 represents daily service. Since the problem does not contain stopping costs, $\dot{\tau}$ and $\ddot{\tau}$ are zero.

Their results for the basic PVRP problem cannot win the best-known results in the literature (by Chao et all and Cordeau et al) for the problem of 100 b data set but they have proved that PVRP with Service Choice could be not only an adaptation but sometimes a necessary option using three scenarios. Scenario I is the traditional PVRP in which nodes must be served at their initially assigned service level, scenario II is PVRP with service choice but only considers the impact of this choice on routing efficiency and the objective function does not include the service benefit term. Scenario III is PVRP with service choice and considers both routing cost and service benefit in the objective function. By default, have been used three benefit parameters a^1=1 \$, $a^2 = 2$ \$, and $a^3 = 5$ \$.

,

4.3 METAHEURISTICS

The meaning of metaheuristics is the method that comes after the heuristic research. So, the main purpose of it is to find a first initial solution by using a heuristic process and improve this to a better one. Over the years, many researchers have worked on this project and have found some very interesting and useful elements even to optimize someone else paper or create a new one.

4.3.1. TABU SEARCH ALGORITHM

One of the most popular methods in literature is the Tabu algorithm. In their paper Cordeau, Laporte and Mercier [10] have presented one version of this algorithm and have applied it to some variants of the VRP. A feature of their approach is the possibility of exploring infeasible solutions and the main purpose is to find the best solution by searching for an initial solution. For the studies about the routing problem with time windows, it is necessary to add some calculations for every point using the lower bound of arrival time (e_i), the upper bound of arrival time (l_i) and the actual arrival time (a_i) of the trucks. The time window constraints are violated, for a point i, if $a_i > l_i$ but its allowed if a truck arrives before e_i. Then the truck waits for time $w_i = e_i - a_i$.

For every solution s ∈ S, let c(s) indicate the total travel time of the routes and let q(s), d(s) and w(s) denote the total violation of load, duration and time window constraints, respectively. Solutions are then evaluated by the function f(s)=c(s)+αq(s)+βd(s)+γw(s), where α, β, γ are nonnegative parameters. This function is not enough to characterize the solution but enter some attributes such as the set B(s) = {(i, r): point i is visited by vehicle r}.

The neighborhood N(s) is defined by replacing the attribute (i, r) with (i, ŕ) where r≠ ŕ. When point i is removed from route r, then the route is simply reconnected by linking the predecessor and successor vertices. Insertion in route ŕ is then performed between two consecutive vertices to minimize the value of f(s). To diversify their algorithm writers proposed a factor $\rho_{i\,k}$, for any $\bar{s} \in N(s)$ such that $f(\bar{s}) \geq f(s)$, which is the number of times attribute (i, r) has been added to the solution during the process. A penalty $p(\bar{s}) = \lambda c(\bar{s})\sqrt{n\,m}\sum_{(i,k)\in B(\bar{s})}\rho_{i\,r}$, $[p(\bar{s}) = 0$ if $f(\bar{s}) < f(s)]$ is added to $f(\bar{s})$, where the quantity $c(\bar{s})\sqrt{n\,|R|}$ (n: number of points, $|R|$:number of vehicles) introduces a correction to adjust the penalties with respect to the total cost and the size of the problem as measured by the number of possible attributes. The parameter λ is used to control the intensity of diversification. These penalties have the effect of driving the process toward less explored regions of the search space whenever a local optimum is reached.

4.3.2 TABU SEARCH ALGORITHM FOR PVRPTW

In accordance with Cordeau, Laporte and Mercier [10], the first step for their solution method is to find an initial solution, which may be infeasible. For a fleet of R vehicles should be founded as routes as the number of vehicles ($|R|$) routes. The process is as follows

- Randomly choose a point j∈ {1, …, n}

- Set r: = 1, where r is a vehicle from the set of vehicles. Writers use the letter k but we will use the letter r for the uniformity of the thesis.

- Using the sequence of customers j, j+1, …, n, 1, …, j-1 and follow the next steps:
  - if the insertion of point i into route r would result in the violation of load or duration constraints, set r: =min {r+1, $|R|$)}

  - insert point i into route r to minimize the increase in the total travel time of route r.
    In that step the insertion of point i can only be performed between successive points j_1 and j_2 such that $e_{j_1} \leq e_i \leq e_{j_2}$ or at the end of the route for satisfying the time window constraints.

The Tabu method uses this initial solution and chooses at each iteration the best non-tabu solution in neighborhood N(s). If the current solution is feasible with respect to load constraints, the values of parameter α, β, γ are divided by 1+δ, where δ is a positive quantity, otherwise they are multiplied by 1+δ, respectively. This process is repeated for some iterations and the best solution s^* is optimized by using the heuristic for the Travelling Salesman Problem with Time Window on every single route.

To be this methodology adapted for the PVRPTW, the first step is defining the set S of solutions formed by routes that satisfy the frequency f_i of the points and the sets S_i of combinations for every point. To construct an initial solution, each point i is assigned a valid visit combination that is randomly chosen in the set S_i. For each day t of the planning period, routes are then constructed by applying the procedure that has been described above but retaining only the points scheduled for visit on day t. The attribute set that is used is B(s)= {(i, r, t): point i is visited by vehicle r on day t}. For every visit combination k ∈ S_i and each day t ∈ $\{1, \dots, T\}$, where k are the visits so the combinations that have to be produced are k-day combinations, let

$$a_{k\,t} = \begin{cases} 1, & \text{if day t belongs to visit combination k} \\ 0, & \text{otherwise} \end{cases}.$$

The following two transformations are allowed during the process:

1. Remove point i from route r of day t and insert it into another route ŕ

2. (a) Replace visit day combination k currently assigned to point i with another combination $\acute{k} \in S_i$.

   (b) For t=1, …, T, do:

- If $a_{k\,t} = 1$ and $a_{\acute{k}\,t} = 0$, remove point i from its route of day t
- If $a_{k\,t} = 0$ and $a_{\acute{k}\,t} = 1$, insert point i into the route of day t minimizing the increase in f(s).

For an initial solution to be found, each customer i is assigned a valid visit combination randomly chosen from the set S_i. For each day t of the period, rotes are then constructed by applying the procedure that authors have produced for the VRP with Time Windows but retaining only the customers scheduled for a visit on day t.

To practice their algorithm, they create new instances by adding random time windows to ten basic PVRP problems including in Cordeau et al paper (Cordeau JF, Gendreau M and Laporte G (1997). A tabu searches heuristic for periodic and multi-depot vehicle routing problems. Networks 30: 105-119.)

4.4 MATHEMATICAL PROGRAMMING BASED APPROACHES

The mathematical programming approach for solving the Periodic Vehicle Routing Problem (PVRP) began to attract significant academic attention in the early to mid-1990s. While the PVRP was first introduced by Beltrami and Bodin in 1981 as we mentioned above, initial solution methods mainly relied on heuristics due to the limited computational capabilities of that era. As optimization techniques evolved, researchers started formulating the PVRP using Mixed Integer Programming (MIP), where both continuous and integer decision variables are optimized simultaneously within a linear framework. MIP models allowed for precise modeling of vehicle routes, customer visit schedules, and operational constraints over a multi-period planning horizon. However, the complexity of the PVRP often led to extremely large models, prompting the development of advanced solution techniques.

In the late 1990s, decomposition methods such as Dantzig-Wolfe decomposition became prominent. This technique involves breaking down a large-scale problem into a master problem and several smaller subproblems, solving them iteratively to improve computational efficiency. Building on this, branch-and-price algorithms combined branch-and-bound tree exploration with column generation (an application of Dantzig-Wolfe decomposition) to dynamically generate only the most promising routing and scheduling variables during the solution process. These techniques enabled the mathematical programming approaches to handle significantly larger and more realistic instances of the PVRP, firmly establishing them as a cornerstone methodology in vehicle routing research.

The implementation of mathematical programming models for solving the Periodic Vehicle Routing Problem (PVRP) has evolved significantly alongside advancements in programming languages and optimization modeling tools. In the early stages, optimization algorithms were often implemented directly in low-level, general-purpose languages such as **Fortran** and **C**, which provided high computational efficiency but required developers to manually manage complex data structures and solver communication. **C** in particular allowed for precise control over memory and algorithmic processes, making it suitable for early custom branch-and-bound and decomposition frameworks, although at the cost of programming complexity.

The late 1980s and 1990s saw the emergence of specialized algebraic modeling languages like **AMPL (A Mathematical Programming Language)** and **LINGO**. **AMPL** offered a powerful, declarative

environment where optimization models could be written in a syntax very close to traditional mathematical notation. It allowed users to define sets, parameters, variables, objectives, and constraints clearly, and to seamlessly link models to solvers such as CPLEX or Gurobi. **LINGO**, developed by LINDO Systems, similarly simplified the modeling and solution of linear, integer, and nonlinear programs, offering an integrated modeling-and-solving environment with a built-in optimization engine. These languages significantly lowered the barrier to implementing and testing large-scale mathematical programming models, including those for PVRP.

As optimization research evolved, **C++** became increasingly popular, particularly from the late 1990s onwards, due to its object-oriented programming features and superior execution speed. **C++** enabled researchers and practitioners to build highly customized and efficient optimization algorithms, such as advanced branch-and-price implementations, metaheuristics, and hybrid frameworks. Its strong performance and flexibility made it the preferred choice for projects where model complexity and solution speed were critical.

In the 2010s, **Python** rapidly gained dominance in the optimization and operations research community. Although slower in raw execution compared to C++, **Python** offers unparalleled ease of use, extensive libraries (such as **PuLP**, **Pyomo**, **OR-Tools**), and direct interfaces with leading solvers. Python's readability, rapid prototyping capabilities, and integration with data analysis, machine learning, and visualization libraries (like Pandas, NumPy, and Matplotlib) make it particularly suited for modern, data-driven optimization approaches to PVRP. Today, Python stands as the primary language for both academic research and practical applications in vehicle routing problems.

This evolution in programming languages and modeling tools has been instrumental in advancing mathematical programming solutions for the PVRP, enabling researchers to model more realistic logistics problems and solve them efficiently.

Parallel to the evolution of programming languages, optimization solvers have also advanced significantly, greatly enhancing the ability to solve mathematical programming models for the Periodic Vehicle Routing Problem (PVRP). Commercial solvers such as **CPLEX** (developed by IBM) and **Gurobi** have become industry standards, offering highly efficient algorithms for a variety of optimization problems. These solvers are particularly powerful in solving **Linear Programming (LP)** problems, where both the objective function and constraints are linear; **Mixed Integer Programming (MIP)** problems, which add integer restrictions to some variables for modeling discrete decisions (like customer assignments); and **Quadratic Programming (QP)** problems, where the objective function is quadratic, but constraints are linear.

CPLEX is well known for its robustness and ability to handle very large MIP models with millions of variables and constraints, making it suitable for complex PVRP formulations. **Gurobi**, on the other hand, has gained a reputation for its cutting-edge performance, multi-threading capabilities, and user-friendly integration with popular modeling languages and environments like Python and AMPL.

In addition to these commercial tools, open-source solvers have played a vital role, especially in academic research and small to medium-scale projects. The **GNU Linear Programming Kit (GLPK)** is an open-source solver capable of handling LP and MIP problems, offering a free alternative for researchers and students. Similarly, **CBC (Coin-or Branch and Cut)**, part of the COIN-OR project, provides a robust open-source solver specifically designed for mixed-integer linear programming,

implementing branch-and-cut algorithms to efficiently explore feasible solutions. The **COIN-OR (Computational Infrastructure for Operations Research)** project is an open-source initiative aimed at providing high-quality, freely available software tools for the operations research and optimization community. Launched in 2000 by IBM researchers, COIN-OR promotes collaborative development of optimization libraries, solvers, and modeling frameworks that are open to everyone. The project hosts a variety of tools ranging from linear and integer programming solvers to modeling languages and network optimization libraries. Solvers such as **CBC (Coin-or Branch and Cut)**, **Clp (Coin-or LP solver)**, and **Ipopt (Interior Point Optimizer)** are part of this initiative, making powerful optimization techniques accessible to academic researchers, students, and practitioners without the need for expensive licenses.

Branch-and-cut is an advanced algorithmic framework used for solving **Mixed Integer Programming (MIP)** problems. It combines two powerful ideas:

- **Branch-and-bound**: A method that systematically explores the solution space by dividing it (branching) and using bounds to prune parts of the search tree that cannot contain better solutions.
- **Cutting planes**: Additional linear inequalities (cuts) that are added to the model to eliminate infeasible parts of the solution space without removing any feasible integer solutions. These cuts tighten the formulation, helping the solver to find integer solutions faster.

In branch-and-cut, the solver alternates between **branching** (splitting the problem into smaller subproblems) and **adding cuts** (strengthening the model) to efficiently home in on the optimal solution. It is the most widely used method in modern solvers for tackling large and complex MIP problems, such as those encountered in the Periodic Vehicle Routing Problem (PVRP).

These solvers, when combined with modern modeling languages and programming environments, enable the effective resolution of increasingly complex PVRP instances, making high-level mathematical optimization accessible to both researchers and practitioners.

In the context of the mathematical approach for solving the PVRP problem, a 5-step algorithm for solving the 100-point problem of the Cristofides and J.E.Beassley paper is developed in the next chapter.

5 SOLUTION ALGORITHM FOR N. CRISTOFIDES AND J.E. BEASSLEY PAPERS DATA SET USING PYTHON

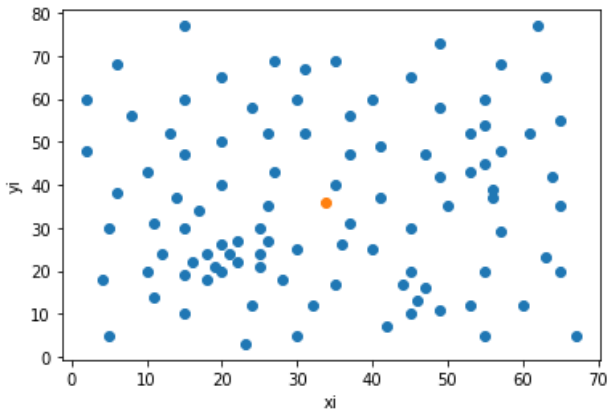
For the 100 customers problem of the Cristofides and J.E.Beassley paper, whose data are listed in the following appendix, a 5-step algorithm was developed to find the optimal assignment of points to 3 salesmans and to find the optimal assignment of the visits required by these points in days as well as the optimal route based on minimizing the total distance.

Step 1: Calculation of gravity point center

The first step is the calculation of the gravity point center of the customers' points. By reading the input data and knowing the geographic coordinates of each customer, the calculation of the gravity point center is trivial. Note that the points are considered as points in the Cartesian plane where the customer's coordinates are translated into Cartesian coordinates. The calculation of the gravity point center is done by calculating: $\bar{x} = \frac{1}{N}\sum_i x_i \quad \bar{y} = \frac{1}{N}\sum_i y_i$

where N=100 is the number of clients and x_i, y_i are the coordinates of each client.

So, schematically, we get the following result:



5-1Scatter Plot of Customers with Gravity Point Center-Step 1

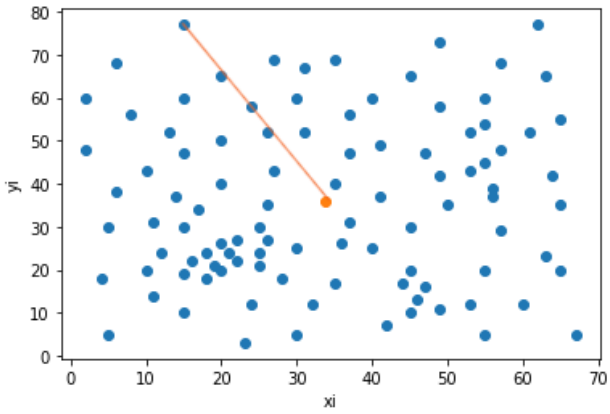
To understand our data, we grouped the points based on their distance from the point's center of gravity into the following 5 categories:

- Category 1: distance from 0 to 10 length units -> 7 points
- Category 2: distance from 10 to 20 length units -> 23 points
- Category 3: distance from 20 to 30 length units -> 33 points
- Category 4: distance from 30 to 40 length units-> 26 points
- Category 5: distance from 40 to 50 length units-> 7 points [67(67,5), 66(49,73), 65(62,77), 64(15,77), 49(6,68), 38(5,5), 35(63,65)]

This categorisation will serve us in the following steps.

Step 2: Ray scanning

The goal in Step 2 is to create initial clusters of customers with the sole purpose, in this step of the algorithm, of evenly distribute the number of visits to each salesman. We pick one of the most distant customers in the grid, which we call the **"initial point"** and create a line starting from the center of gravity of the point cloud and extending in the direction of that point. Thus, we create the following line in the reference example:

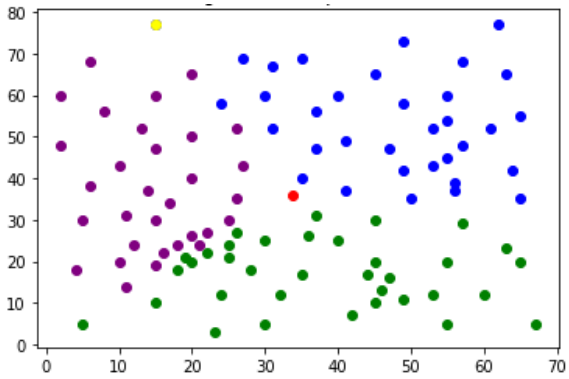


5-2Scatter Plot of Customers with Ray Scanning -Step 2

We consider this line as the **reference axis** which for each client defines an angle ϕ with respect to it. Obviously, the **"initial point"** has ϕ=0 while all other customers are sorted in ascending order based on their angle φ_i. To define the area of responsibility for the first salesman, we mentally rotate the reference axis from the position in which it is located clockwise by assigning customers in the current area of responsibility until the number of visits of all customers in the area is equal to the average number of visits, we would like each salesman to have. In our problem this number is equal to $\frac{\sum_i u_i}{\text{saleman }(3)} = 58.$

Once the number of customers visits contained between the initial and final radius of the scanner, we created becomes perhaps equal to the average number of visits, we stop creating the current responsibility area, thus defining the responsibility area of the first salesman. The above procedure is repeated as many times as the number of salesmen (in our problem is 3) in order to define for each of them their area of responsibility.

Applying this procedure to the reference example, the following three areas of responsibility are created schematically:



5-3 Clustering step 3 with initial point 64 (example)

The center of gravity of the points is shown in red, the initial point is shown in yellow, the first salesman area of responsibility is shown in blue, the second salesman area of responsibility is shown in green, and the third salesman area of responsibility is shown in purple.

At the end of this step, initial areas of responsibility have been defined based only on the number of visits. It is possible, that if the customer orders in each area of responsibility are aggregated and the turnover is not evenly distributed among the salesman as we would like. To get around this accidental mismatch, the next Step 3 was developed.

Step 3: Problem of assigning customers to salesman

The previous Step 2 gave us an initial assignment of customers to salesmen, but this may be problematic in terms of the number of orders each salesman disburses.

To solve this problem, a mixed integer mathematical programming model was developed that aims to transfer customers from one salesman to another salesman in order to achieve the best possible turnover for each salesman.

We decide that the cost of assigning customer i to salesman k is 0 if the customer is eventually assigned to the salesman originally assigned in the previous Step 2. If he is assigned to any other salesman, then the cost is equal to the distance between the customer's point and the gravity center of the other clusters.

Data	Definition
i	Customer
k	Salesman
v_V	Margin of variation in the number of visits between salesmen
v_T	Margin of variation of orders/turnover between salesmen
q_i	Order/Customer's I turnover
$\bar{Q}$	Average order size
$\bar{V}$	Average number of visits
u_i	Required frequency of customer's i visits
c_{ik}	Cost of assigning customer i to salesman k and is 0 if the customer is eventually assigned to the salesman originally assigned in Step 2 and is equal

	to the distance between the customer's point and the gravity center of the other clusters.

Table 1Problem of assigning customers to salesman-step 3

Definition of model

Decision variable: $x_{ik} = \begin{cases} 1, \ when\ customer\ i\ is\ assigned\ to\ salesman\ k \\ 0, \ otherwise \end{cases}$

(5.1) Min $\sum_k c_{ik} x_{ik}$

(5.2) $\sum_k x_{ik} = 1 \ \forall \ i$

(5.3) $v_T \bar{Q} \leq \ \sum_i x_{ik} q_i u_i \leq (2 - v_T)\overline{Q} \ \forall k$

(5.4) $v_V \bar{V} \leq \ \sum_i x_{ik} u_i \leq (2 - v_V)\overline{V} \ \forall k$

FORMULATION 7 PROBLEM OF ASSIGNING CUSTOMERS TO SALESMEN -STEP 3

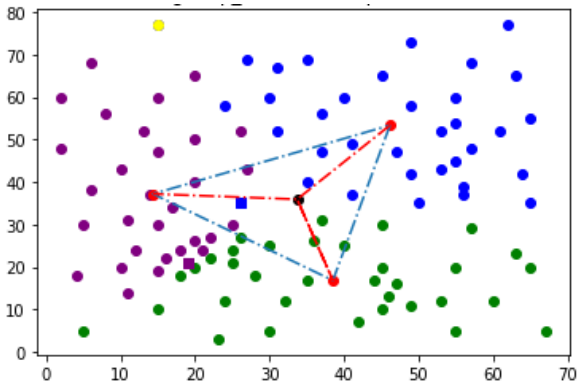
Function (5.1) corresponds to the objective function of the model and aims to minimize the cost of (re)assigning customers to salesmen. Function (5.2) guarantees that each customer will be assigned to exactly one salesman. Function (5.3) controls that the turnover disbursed by each salesman after assignment should not be outside the prescribed turnover limits and function (5.4) controls that the number of visits will not violate the respective limits either.

By solving the above model, customers have been assigned to salesmen in such a way that two conditions are met which may not have been met in the previous step: a) the number of visits that each salesman will be asked to make throughout the time horizon is not less than 95% of the average and at the same time b) the turnover disbursed from the orders by each salesman is not less than 95% of the average among salespeople.

In addition, the algorithm maximizes the geographic distribution of customers across salesmen by minimizing the number of those customers who will be assigned to clusters different from those assigned in Step 2.

In case the cost of the solution returned to us is zero, this means that there is no client assigned to another cluster beyond the one originally assigned in Step 2. Otherwise, two cases may have occurred: a) geographical overlaps will occur which we will correct in the next Step 4 of the algorithm and/or b) those customers who were assigned to another cluster were in the cluster boundaries and thus no overlap was created.

Applying this procedure to the reference example, the following three areas of responsibility are created schematically with the points that have ready transpositions being represented by a square:



5-4 Clustering step 3 with initial point=64 (example)

Step 4: Problem transferring customers from salesman to salesman

This step shall be triggered when overlapping areas of responsibility are detected in the previous step. We assume that each customer makes a transfer request from one salesman to another salesman due to being a customer of one salesman in another salesman's area of responsibility.

In order to find any overlaps, a comparison is made between the mapped clients from Step 3 and Step 2. Those customers who are in Step 3 in the responsibility area different from the one they were in Step 2 are the ones who generate the transfer requests. A transfer request is the transfer of a customer who was placed in a different responsibility area to its original area. This procedure creates for example the following transfer table:

Customer	Salesman		Request
	From (step 3) Δ^-	To (step 2) Δ^+	
5	0	2	False
18	0	3	False
187	0	1	False
…	…	…	…

Where each customer requests a transfer from one salesman to another and the last column shows the state in which this request is found after solving the following mathematical model.

For the following mathematical model, we introduce the binary variable z_s where s is the request (i.e., the row of the above table) with a value of 1 if the request is eventually accepted and 0 if not, $z_s \in \{0,1\}$

Data	Definition
s	Index of a transfer request containing customer i and salesman from-to
k	Index of salesman
Q_{min}/Q_{max}	Minimum and maximum turnover
V_{min}/V_{max}	Minimum and maximum number of visits
$\boldsymbol{\Delta}^{+}$	Total requests concerning customers assigned to a salesman (column "To" in the table above)
$\boldsymbol{\Delta}^{-}$	Total requests concerning customers who cease to be assigned to a salesman (column 'From ' in the above table)
Q_k	Total turnover generated for each salesman in the previous Step 3
V_k	Total number of visits generated for each vendor in the previous Step 3
q_s	Quantity corresponding to the customer requesting a transfer from one salesman to another and corresponding to request s.
f_s	Visit frequency corresponding to the customer requesting a transfer from one salesman to another and corresponding to request s.

Table 2 Problem transferring customers from salesman to salesman -step 4

To solve this problem, we construct the following mixed-integer linear programming model:

$$(5.5)\quad \mathrm{Max}\sum_{\mathrm{i}} \mathrm{z_s}$$

$$(5.6)\quad \mathrm{Q_{min}} - \mathrm{Q_k} \le \sum_{\mathrm{s}\in\Delta^{+}} \mathrm{z_s q_s f_s} - \sum_{\mathrm{s}\in\Delta^{-}} \mathrm{z_s q_s f_s} \le \mathrm{Q_{max}} - \mathrm{Q_k}\ \forall \mathrm{k}$$

$$(5.7)\quad \mathrm{V_{min}} - \mathrm{V_k} \le \sum_{\mathrm{s}\in\Delta^{+}} \mathrm{z_s f_s} - \sum_{\mathrm{i}\in\Delta^{-}} \mathrm{z_s f_s} \le \mathrm{V_{max}} - \mathrm{V_k}\ \forall \mathrm{k}$$

FORMULATION 8 PROBLEM OF TRANSFERING CUSTOMERS FROM SALESMAN TO SALESMAN - STEP 4

The function (5.5) is the objective function of the mathematical model which seeks to maximize the satisfaction of the transfer requests. The functions (5.6) and (5.7) guarantee that the quantities of orders (turnover) and the number of visits generated after the transfers must meet the thresholds we have defined for each salesman.

At this point the algorithm checks whether or not all the requests have been satisfied. If not, we have two options, depending on the user's preference:

- If the user has indicated that they want absolute geographic distribution, then we satisfy the requests of unsatisfied customers at the expense of equal distribution of turnover and total number of visits,
- If the user does not want absolute geographic allocation, then we leave customers as they are while respecting the constraints on turnover and number of visits by making a concession on geographic allocation.

Step 5: Daily assignment of customers to salesmen

We already know which customer belongs to which salesman based on our three main criteria (geographic distribution, turnover and number of visits), but we don't know the day on which they will be assigned. The purpose of this step is to assign customers to days based on each customer's desired frequency and geographic distribution.

Ideally, we would like each day of the salesman's commute to see as many nearby customers as possible in the area they are visiting. To achieve this, we classify each salesman's customers into clusters and ask them to visit at most one cluster in their daily schedules. To create the clusters, we use the k-means algorithm. Each cluster is associated with a centroid (center) and each point is assigned to the cluster with the closest center. The number of clusters (k) has to be determined from the beginning. We used the Elbow Algorithm to find the optimal number of clusters k. Basically the program iteratively runs the algorithm for k=10, and for each k the WCSS (within cluster sum of squares) index is calculated which measures how well the data is clustered around the corresponding center of gravity. For our data it follows that the number k should be equal to 5.

But so far, we have not considered the frequency of visits. In order not to introduce the concept of periodicity in our TSP we propose to follow a well-known method in literature, according to which we create virtual points with the same coordinates for those points that require multiple visits within our period. This way instead of having one point requiring 2 or 3 visits we will have 2 or 3 points requiring one visit.

After creating these points, we will apply the k-means algorithm requiring points with the same coordinates to be placed in different clusters. This way we will be able to perceive each cluster as a day and then apply one of the well-known algorithms in the literature for TSP to derive the optimal paths.

In this paper the following Miller – Tucker- Zemlin formulation was used for the TSP problem however any formatting of the literature could be used. For this formulation we calculate the distance of each point i from point j within each cluster (d_{ij}) and we define the following variables.

Decision Variables:

$$x_{ij} = \begin{cases} 1 \text{ , if salesman goes from customer i to customer j} \\ 0, \text{otherwise} \end{cases}$$

y_i positive continuous variable for the sub tour elimination constraints

Definition of the model:

(5.8) Min $\sum_{i\ j} d_{ij} x_{ij}$

(5.9) $\sum_i x_{ij} = 1,\ \forall j\ ,\ i \neq j$

(5.10) $\sum_j x_{ij} = 1,\ \forall i\ ,\ i \neq j$

(5.11) $y_i - (n+1)x_{ij} \geq y_j - n\ , \forall\ i\ , i \neq 0, j \neq 0, i$

FORMULATION 9 TSP -STEP 5

Equation 5.8 is the objective function of the model and minimizes the distance that the seller has to travel in a day. Equation 5.9 and 5.10 guarantees that if a seller goes to point i, he will also leave from that point. Equation 5.11 is called the **subtour elimination constraints** and essentially eliminates the creation of shorter paths within the total path that the salesman must follow. If we did not put this constraint (or some other form of constraint developed in literature), the optimal solution might be to trap the salesman between two or three points.

With the completion of all the above steps we have developed a solution for the optimal assignment of 100 customers to 3 salesmen based on the geographical distribution and equal distribution of turnover and the optimal route that each salesperson should follow each day of the week.

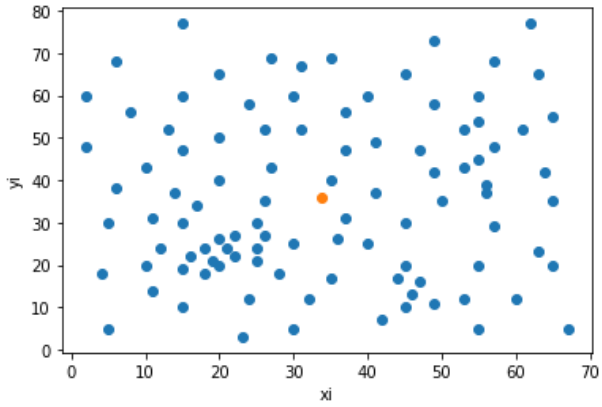
The following appendix analyses the program developed using the Python interface of the Gurobi optimization software in the Jupyter Notebook environment. Python is a high-level programming language celebrated for its simplicity, readability, and versatility. Furthermore, it supports multiple programming paradigms and boasts an extensive standard library that simplifies coding tasks. For those who are new to operations research, the Gurobi site offers solved optimization examples so that the developer can become familiar with the interface and the way of coding.

6 APPLICATION OF THE ALGORITHM OF CHAPTER 5 BY SELECTING AS INITIAL POINTS THE 7 MOST DISTANT POINTS OF CUSTOMER

For the best possible distribution of points into clusters based on their position on the Cartesian plane, we calculate the following data for the most distant points of our customer cloud from the center of gravity using the categorization we made in the previous chapter in step 1.

6.1 DAILY ROUTES AND DISTANCES FOR THE INITIAL POINT I=65

1. Step 1: Calculation of gravity point center

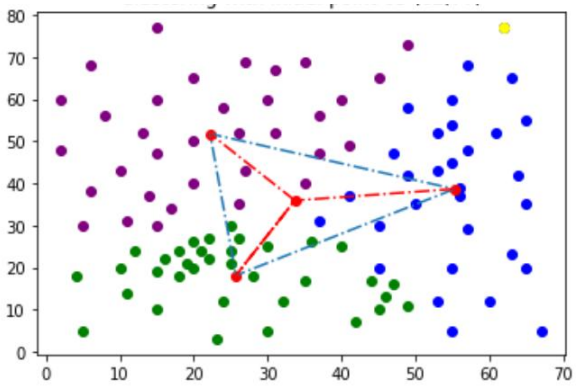


6-1 Scatter Plot of Customers with gravity point center and initial point i=65

2. Step 2: Ray scanning

The furthest point from the center of gravity of the customer's points is i=65, $x_i = 62$, $y_i = 77$with distance equal to 49.884430436760525 distance units

Clustering based on geographical distribution, so every salesman would have 58 number of visits.

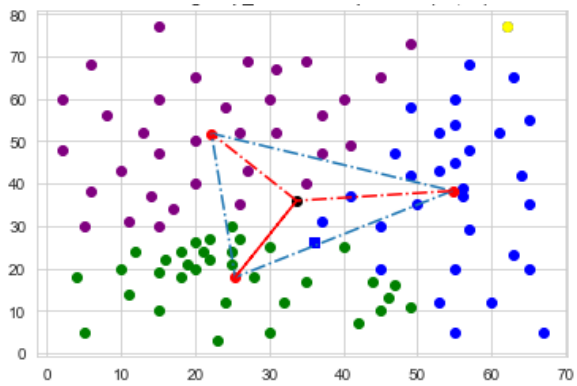


6-2: clustering step2 with initial point i=65

Points in blue are those that have been assigned to the first salesman, those in green to the second and those to purple to the third. Points in red are the centers of gravity for all points and for each cluster separately and the point in yellow is the initial point i=65.

3. Step 3: Problem of assigning customers to salesman

The solution to the problem in step 3 showed that client 58 is moved from cluster 2 (green) in step 2 to cluster 1 (blue) in step 3. The next graph shows the ne clustering based on turnover with the point that changed salesman being in a blue square.



6-3Clustering step3 with initial point i=65

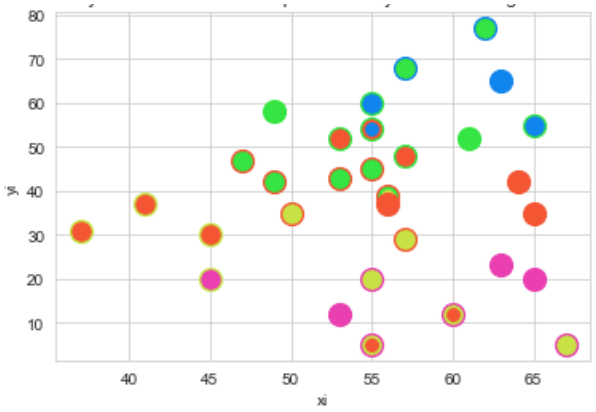
4. Step 4: Problem transferring customers from salesman to salesman

The solution to the problem in step 4 resulted in the optimal solution that point 58 cannot return to its original clustering of step 2, i.e., the client's relocation request was not accepted. So in this case we chose to continue with the cluster of step 2

5. Step 5: Daily assignment of customers to salesmen

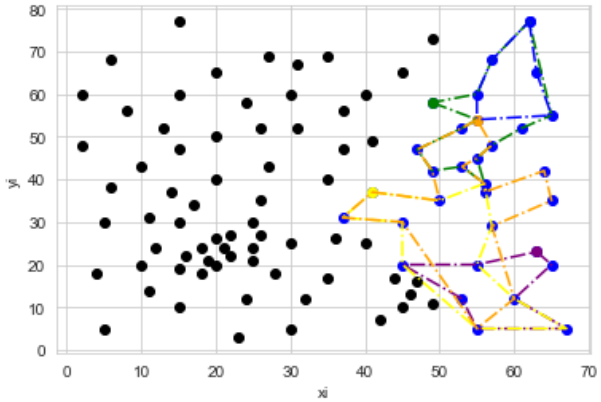
We first divide the assigned points into 5 clusters based on what has been mentioned in step 5. From the program we get the following results for each salesman separately.

Salesman 1 (Blue):



6-4Daily Visits of salesman 1

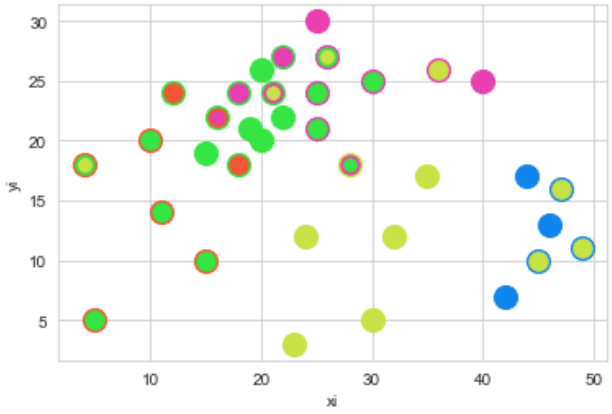
Days	Number of Points	Total Distance of the route for the day	Points of Route of the day for salesman 1 (Blue)
Green	14	99.774	[49,58] [55,54] [53,52] [47,47] [49,42] [53,43] [56,39] [55,45] [57,48] [65,55] [62,77] [55,60] [49,58] [62,77] [55,60] [49,58]
Green	8	72.077	[63,23] [65,20] [60,12] [67,5] [55,5] [53,12] [45,20] [55,20] [63,23]]
Yellow	11	110.335	[41,37] [50,35] [56,39] [57,29] [55,20] [60,12] [67,5] [55,5] [45,20] [45,3] [37,31] [41,37]
Blue	6	56.832	[62,77] [63,65] [65,55] [55,54] [55,60] [57,68] [62,77]
Orange	18	146.642	[55,54] [53,52] [47,47] [49,42] [50,35] [41,37] [37,31] [45,30] [55,5] [60,12] [57,29] [65,35] [64,42] [56,37] [56,39] [53,43] [55,45] [57,48] [55,54]



6-5 Scatter Plot of salesmans 1 visits in one period

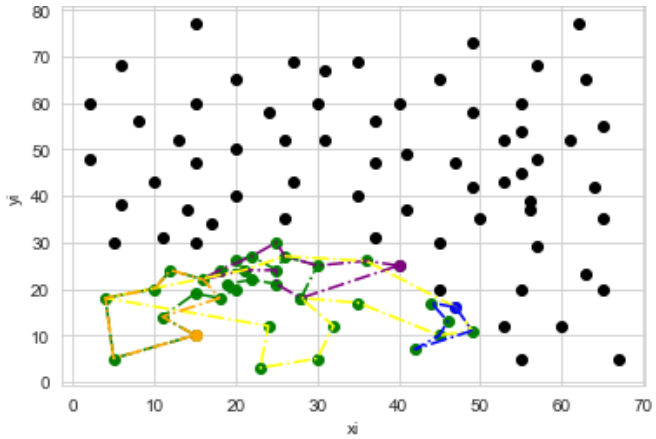
TOTAL DISTANCE OF SALESMAN 1 (BLUE): 485.66 distance units.

Salesman 2 (Green):



6-6Daily Visits of salesman 2

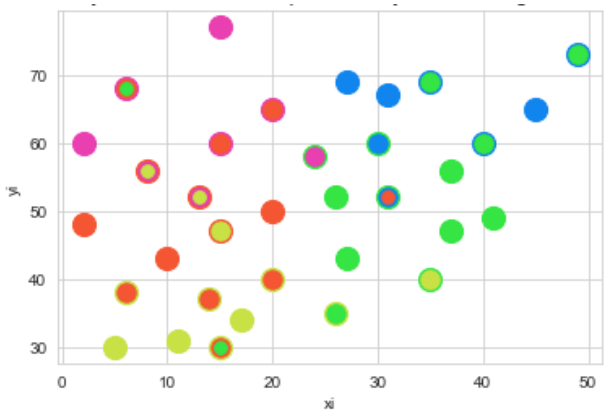
Days	Number of Points	Total Distance of the route for the day	Points of Route of the day for salesman 2(Green)
Green	21	98.9751	[28,18] [30,25] [26,27] [25,24] [22,27] [20,26] [21,24] [8,24] [16,22] [12,24] [10,20] [4,18] [5,5] [15,10] [11,14] [15,19] [18,18] [20,20] [19,21] [22,22] [25,21]
Green	12	60.431	[40,25] [36,26] [30,25] [26,27] [25,30] [22,27] [18,24] [16,22] [21,24] [25,24] [25,21] [28,18] [40,25]
Yellow	13	129.268	[7,26] [36,26] [26,27] [21,24] [4,18] [24,12] [23,3] [30,5] [32,12] [28,18] [35,17] [45,10] [49,11] [47,16]
Blue	6	28.486	[47,16] [44,17] [46,13] [45,10] [42,7] [49,11]
Orange	8	57.678	[15,10] [11,14] [18,18] [16,22] [12,24] [10,20] [4,18] [5,5] [15,10]



6-7Scatter Plot of salesman 2 visits in one period

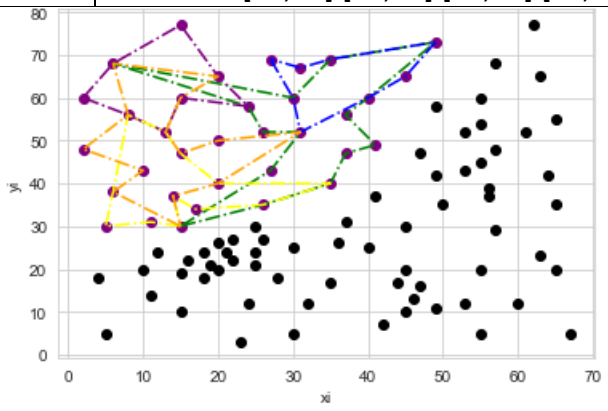
TOTAL DISTANCE OF SALESMAN 2 (GREEN): 374.838 distance units.

Salesman 3 (Purple):



6-8Daily Visits od salesman3

Days	Number of Points	Total Distance of the route for the day	Points of Route of the day for salesman 3 (Purple)
Green	15	172.615	[15,30] [26,35] [35,40] [37,47] [41,49] [37,56] [40,60] [49,73] [35,69] [30,60] [6,68] [24,58] [26,52] [31,52] [27,43] [15,30]
Green	8	73.814	[2,60] [6,68] [15,77] [20,65] [24,58] [15,60] [13,52] [8,56] [2,60]
Yellow	12	102.434	[15,30] [11,31] [5,30] [6,38] [8,56] [13,52] [15,47] [20,40] [35,40] [26,35] [17,34] [14,37] [15,30]
Blue	8	69.11	[27,69] [30,0] [31,52] [40,60] [45,65] [49,73] [35,69] [31,67] [27,69]
Orange	14	132.134	[15,30] [6,38] [10,43] [2,48] [8,56] [6,68] [20,65] [15,60] [13,52] [15,47] [20,50] [31,52] [20,40] [14,37] [15,30]

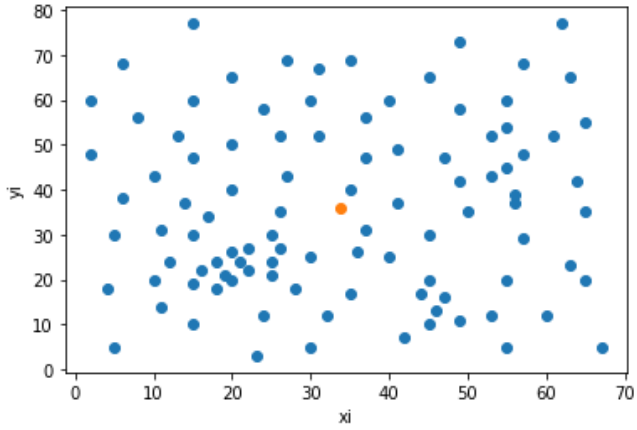


6-9Scatter Plot of salesman 3 visits in one period

TOTAL DISTANCE OF SALESMAN 3 (PURPLE): 550.107 distance units.

6.2 DAILY ROUTES AND DISTANCES FOR THE INITIAL POINT I= 67

1. Step 1: Calculation of gravity point center

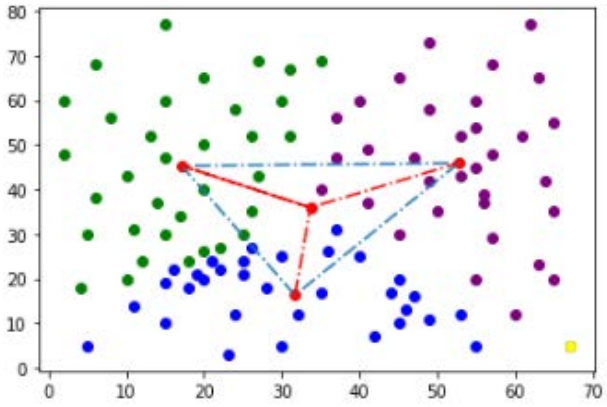


6-10Scatter Plot of Customers with gravity point center and initial point i=67

2. Step 2: Ray scanning

The furthest point from the center of gravity of the customer's points is i=67, $x_i = 67$, $y_i = 5$ with distance equal to 45.44157127565023 distance units.

Clustering based on geographical distribution, so every salesman would have 58 number of visits.

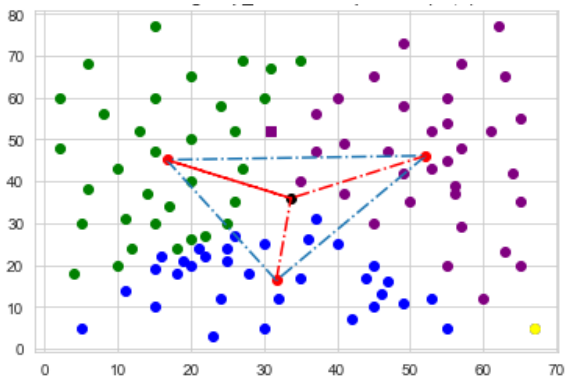


6-11Clustering step 2 with initial point i=67

Points in blue are those that have been assigned to the first salesman, those in green to the second and those to purple to the third. Points in red are the centers of gravity for all points and for each cluster separately and the point in yellow is the initial point i=67.

3. Step 3: Problem of assigning customers to salesman

The solution to the problem in step 3 showed that client 31 is moved from cluster 2 (green) in step 2 to cluster 3 (purple) in step 3. The next graph shows the ne clustering based on turnover with the point that changed salesman being in a blue square.



6-12Clustering step3 with initial point i=67

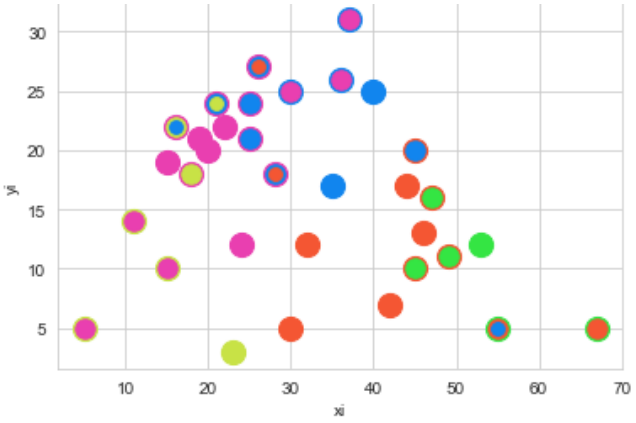
4. Step 4: Problem transferring customers from salesman to salesman

The solution to the problem in step 4 resulted in the optimal solution that point 31 cannot return to its original clustering of step 2, i.e., the client's relocation request was not accepted. So in this case we chose to continue with the cluster of step 2

5. Step 5: Daily assignment of customers to salesmen

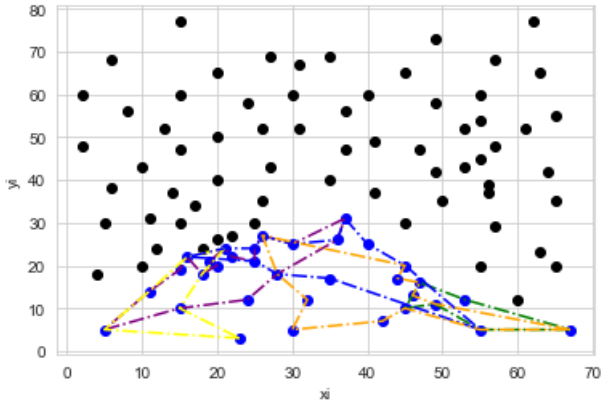
We first divide the assigned points into 5 clusters based on what has been mentioned in step 5. From the program we get the following results for each seller separately.

Salesman 1 (Blue):



6-13Daily Visits od salesman 1

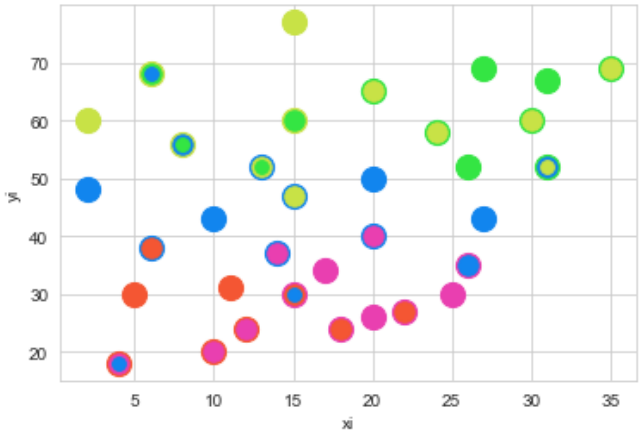
Days	Number of Points	Total Distance of the route for the day	Points of Route of the day for salesman 1 (Blue)
Green	6	53.796	[67,5] [55,5] [49,11] [45,10] [47,16] [53,12] [67,5]
Green	18	101.978	[37,31] [30,25] [26,27] [25,24] [25,21] [22,22] [21,24] [19,21] [20,20] [18,18] [16,22] [15,19] [11,14] [5,5] [15,10] [24,12] [28,18] [36,26 37,31]
Yellow	7	69.629	[23,3] [5,5] [11,14 [16,22] [21,24] [18,18] [15,10] [23,3]
Blue	13	103.701	45,20] [55,5] [35,17] [28,18] [25,21] [16,22] [21,24] [25,24] [26,27] [30,25] [36,26] [37,31] [40,25] [45,20]
Orange	13	115.614	[67,5] [49,11] [46,13] [47,16] [44,17] [45,20] [26,27] [28,18] [32,12] [30,5] [42,7] [45,10] [55,5] [67,5]



6-14Scatter Plot of salesman 1 visits in one period

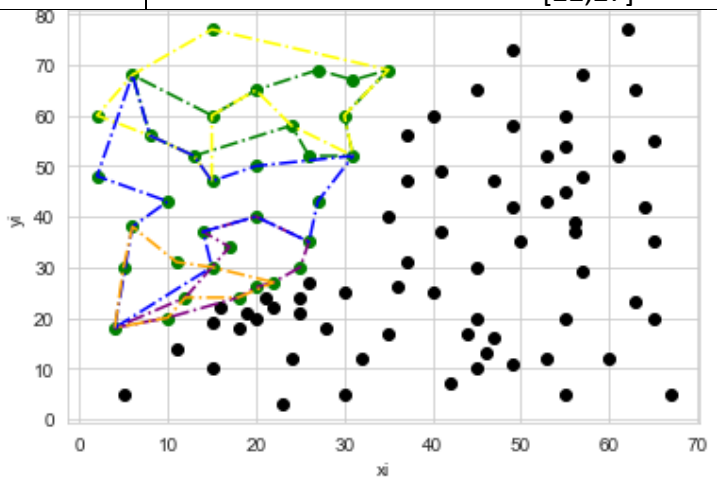
TOTAL DISTANCE OF SALESMAN 1 (BLUE): 444.718 distance units.

Salesman 2 (Green):



6-15Daily Visits od salesman 2

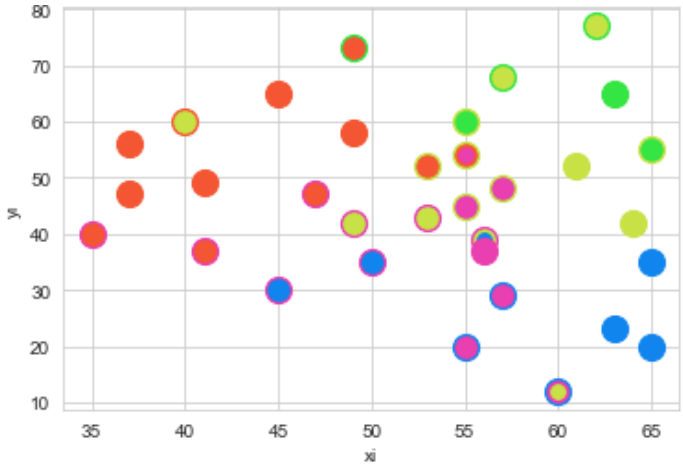
Days	Number of Points	Total Distance of the route for the day	Points of Route of the day for salesman 2(Green)
Green	12	96.901	[13,52] [8,56] [6,68] [15,60] [20,65] [27,69] [31,69] [35,69] [30,60] [31,52] [26,52] [24,58] [13,52]
Green	12	69.6161	[22,27] [20,26] [18,24] [10,20] [4,18] [12,24] [15,30] [17,34] [14,37] [20,40] [26,35] [25,30] [22,27]
Yellow	12	117.923	[15,47] [15,60] [20,65] [24,58] [31,52] [30,60] [35,69] [15,77] [6,68] [2,60] [8,56] [13,52] [15,47]
Blue	15	153.077	[4,18] [6,38] [10,43] [2,48] [6,68] [8,56] [13,52] [15,47] [20,50] [31,52] [27,43] [26,35] [20,40] [14,37] [15,30] [4,18]
Orange	9	62.2421	[22,27] [15,30] [11,31] [6,38] [5,30] [4,18] [10,20] [12,24] [18,24] [22,27]



6-16Scatter Plot of salesman 2 visits in one period

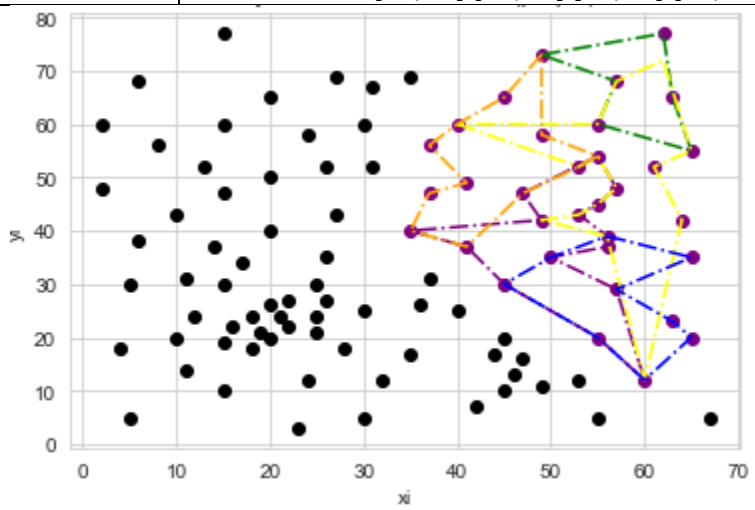
TOTAL DISTANCE OF SALESMAN 2 (GREEN): 499.7592 distance units.

Salesman 3 (Purple):



6-17Daily Visits of salesman3

Days	Number of Points	Total Distance of the route for the day	Points of Route of the day for salesman 3 (Purple)
Green	6	64.701	[49,73] [62,77] [63,65] [65,55] [55,60] [57,68] [49,73]
Green	15	121.07	[35,40] [41,37] [45,30] [55,20] [60,12] [57,29] [60,35] [56,37] [56,39] [53,43] [55,45] [57,48] [55,54] [47,47] [49,42] [35,40]
Yellow	12	171.3356	[40,60] [53,52] [55,54] [7,48] [55,45] [53,43] [49,42] [56,39] [60,12] [64,42] [61,52] [65,55] [62,72] [57,68] [55,60] [40,60]
Blue	9	79.232	[56,39] [65,35] [57,29] [63,23] [65,20] [60,12] [55,20] [45,3] [50,35] [56,39]
Orange	12	92.048	[37,56] [40,60] [45,65] [49,73] [49,58] [55,54] [53,52] [47,47] [41,37] [35,40] [37,47] [41,49] [37,56]

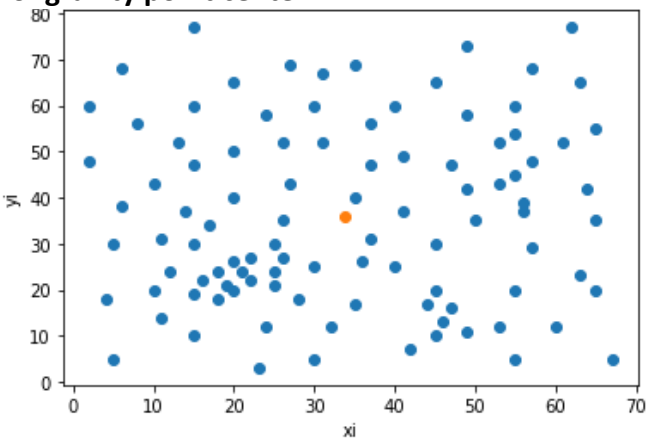


6-18Scatter Plot of salesman 3 visits in one period

TOTAL DISTANCE OF SALESMAN 3 (PURPLE): 528.3866 distance units.

6.3 DAILY ROUTES AND DISTANCES FOR THE INITIAL POINT I= 64

1. **Step 1: Calculation of gravity point center**

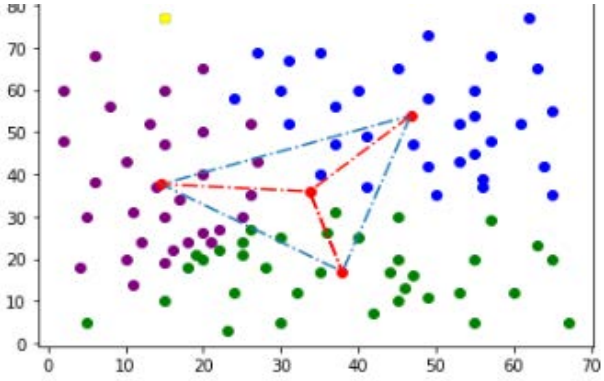


6-19Scatter Plot of Customers with gravity point center and initial point i=64

2. **Step 2: Ray scanning**

The furthest point from the center of gravity of the customer's points is i=64, $x_i = 15$, $y_i = 77$ with distance equal to 45.13597678127726 45.44157127565023 distance units.

Clustering based on geographical distribution, so every salesman would have 58 number of visits.

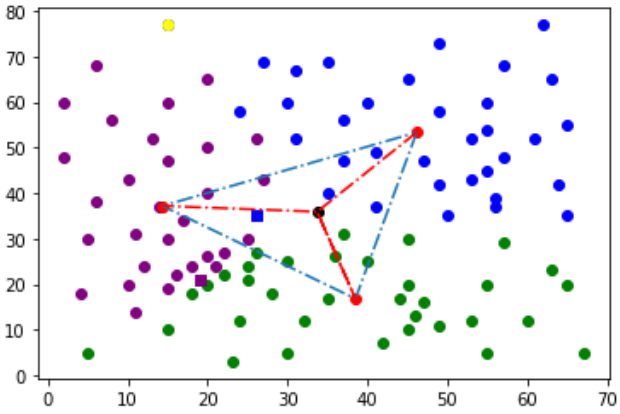


6-20Clustering step 2 with initial point i=64

Points in blue are those that have been assigned to the first salesman, those in green to the second and those to purple to the third. Points in red are the centers of gravity for all points and for each cluster separately and the point in yellow is the initial point i=64.

3. **Step 3: Problem of assigning customers to salesman**

The solution to the problem in step 3 showed that client 89 is moved from cluster 3 (purple) in step 2 to cluster 1 (blue) in step 3. Also, client 98 is moved from cluster 2 (green) in step 2 to cluster 3 (purple) in step 3. The next graph shows the new clustering based on turnover with the point that changed salesman being in a square.



6-21Clustering step3 with initial point i=64

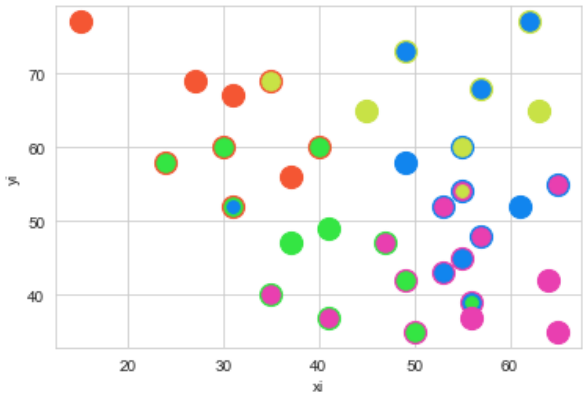
4. Step 4: Problem transferring customers from salesman to salesman

The solution to the problem in step 4 resulted in the optimal solution that point 31 cannot return to its original clustering of step 2, i.e., the client's relocation request was not accepted. So in this case we chose to continue with the cluster of step 2

5. Step 5: Daily assignment of customers to salesmen

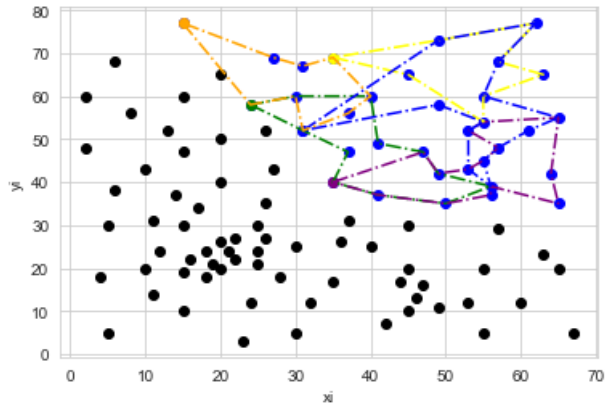
We first divide the assigned points into 5 clusters based on what has been mentioned in step 5. From the program we get the following results for each seller separately.

Salesman 1 (Blue):



6-22Daily Visits of salesman 1

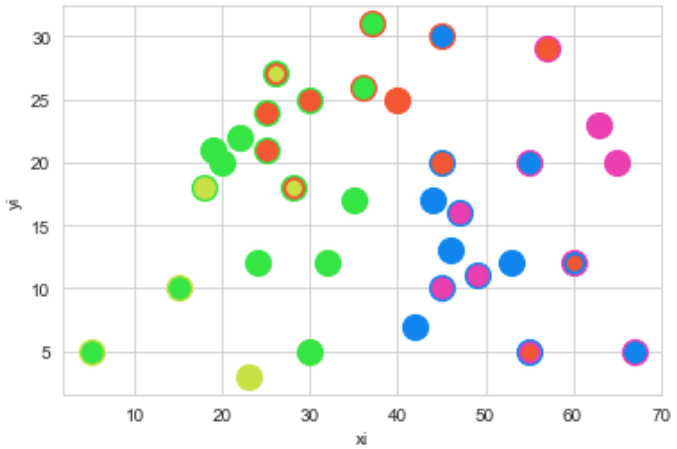
Days	Number of Points	Total Distance of the route for the day	Points of Route of the day for salesman 1 (Blue)
Green	12	94.145	[24, 58] [31, 52] [37, 47] [35,40] [41,37] [50,35] [56,39] [49,42] [47,47] [41,49] [40,60] [30,60] [24,58]
Green	15	102.58	[35,40] [47,47] [49,42] [53,43] [55,45] [57,48] [53,52] [55,54] [65,55] [64,42] [65,35] [56,39] [56,37] [50,35] [41,37] [35,40]
Yellow	8	86.235	[35,69] [45,65] [55,54] [55,60] [63,65] [57,68] [62,77] [49,73] [35,69]
Blue	14	134.341	[31,52] [49,73] [62,77] [57,68] [55,60] [65,55] [61,52] [57,48] [55,45] [56,39] [53,43] [53,52] [55,54] [49,58] [31,52]
Orange	9	81.284	[15,77] [24,58] [30,60] [31,52] [37,56] [40,60] [35,69] [31,67] [27,69] [15,77]



6-23Scatter Plot of salesman 1 visits in one period

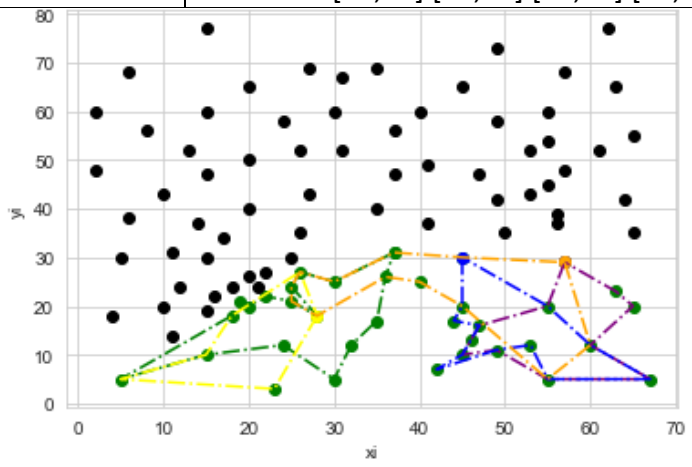
TOTAL DISTANCE OF SALESMAN 1 (BLUE): 498.585 distance units.

Salesman 2 (Green):



6-24Daily Visits of salesman 2

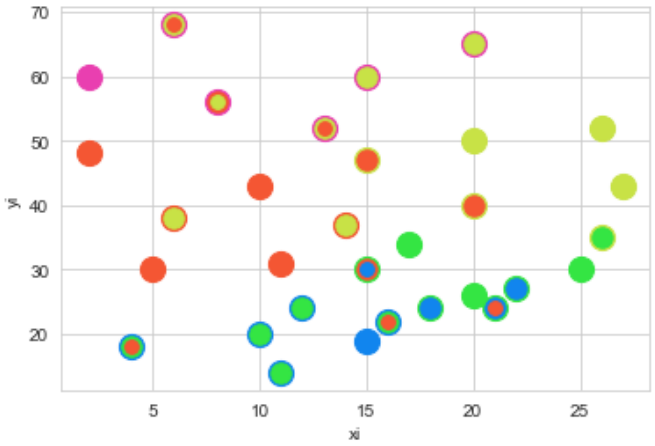
Days	Number of Points	Total Distance of the route for the day	Points of Route of the day for salesman 2(Green)
Green	17	113.641	[37,31] [30,25] [26,27] [25,24] [28,18] [25,21] [22,22] [20,20] [19,21] [18,18] [5,5] [15,10] [24,12] [30,5] [32,12] [35,17] [36,26] [37,31]
Green	10	80.521	[57,29] [55,20] [47,16] [45,10] [49,11] [55,5] [67,5] [60,12] [65,20] [63,23] [57,29]
Yellow	6	74.908	[28,18] [23,3] [5,5] [15,10] [18,18] [26,27] [28,18]
Blue	13	91.831	[45,30] [45,20] [44,17][47,16] [46,13] [45,10] [42,7] [49,11] [53,12] [55,5] [67,5] [60,12] [55,20] [45,30]
Orange	13	110.602	[57,29] [60,12] [55,5] [45,20] [40,25] [36,26] [28,18] [25,21] [25,24] [26,27] [30,25] [37,31] [45,30] [57,29]



6-25Scatter Plot of salesman 2 visits in one period

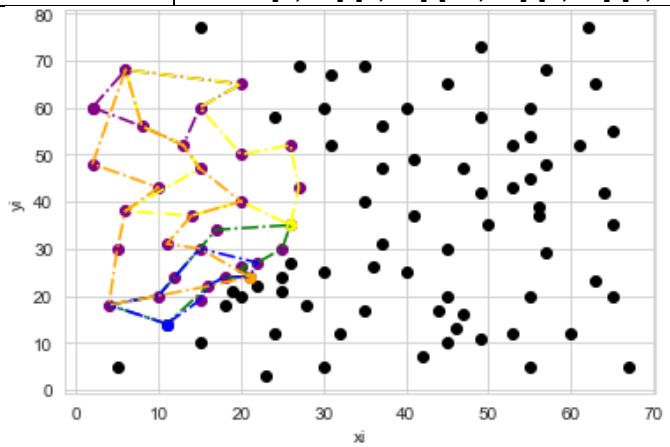
TOTAL DISTANCE OF SALESMAN 2 (GREEN): 471.503 distance units.

Salesman 3 (Purple):



6-26Daily Visits of salesman 3

Days	Number of Points	Total Distance of the route for the day	Points of Route of the day for salesman 3 (Purple)
Green	13	68.17	[11,14] [4,18] [10,20] [12,24] [15,30] [17,34] [26,35] [35,30] [22,27] [20,26] [21,24] [18,24] [16,22] [11,14]
Green	6	52.193	[2,60] [8,56] [13,52] [15,60] [20,65] [6,68] [2,60]
Yellow	13	115.273	[26,35] [27,43] [26,52] [20,50] [15,60] [20,65] [6,68] [8,56] [13,52] [15,47] [6,38] [14,37] [20,40] [26,35]
0Blue	10	51.738	[11,14] [15,19] [16,22] [18,24] [21,24] [22,27] [15,30] [12,24] [10,20] [4,18] [11,14]
Orange	15	132.951	[21,24] [15,30] [11,31] [14,37] [20,40] [15,47] [13,52] [8,56] [6,68] [2,48] [10,43] [6,38] [5,30] [4,18] [16,2] [21,24]

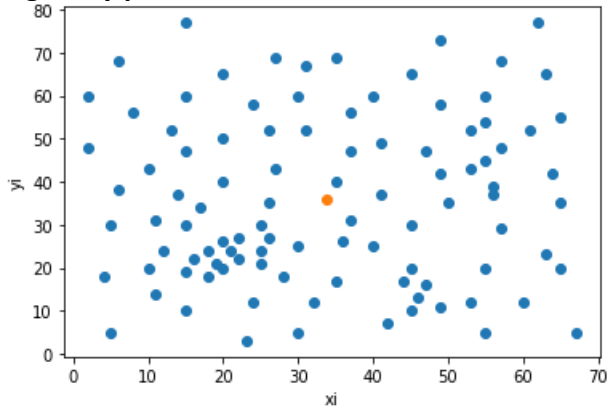


6-27Scatter Plot of salesman 3 visits in one period

TOTAL DISTANCE OF SALESMAN 3 (PURPLE): 420.325 distance units.

6.4 DAILY ROUTES AND DISTANCES FOR THE INITIAL POINT I= 49

1. **Step 1: Calculation of gravity point center**

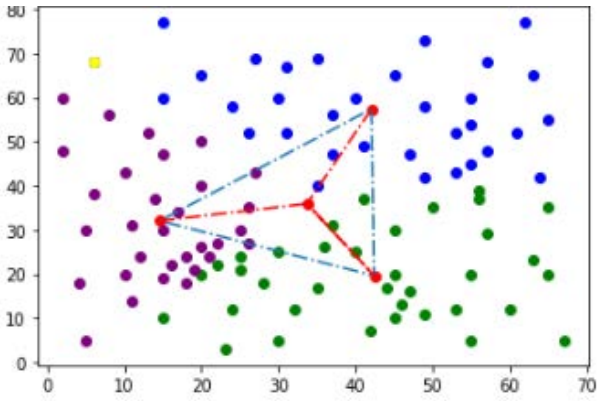


6-28Scatter Plot of Customers with gravity point center and initial point i=49

2. **Step 2: Ray scanning**

The furthest point from the center of gravity of the customer's points is i=49, $x_i = 6$, $y_i = 68$ with distance equal to 42.38415269885668 distance units.

Clustering based on geographical distribution, so every salesman would have 58 number of visits.

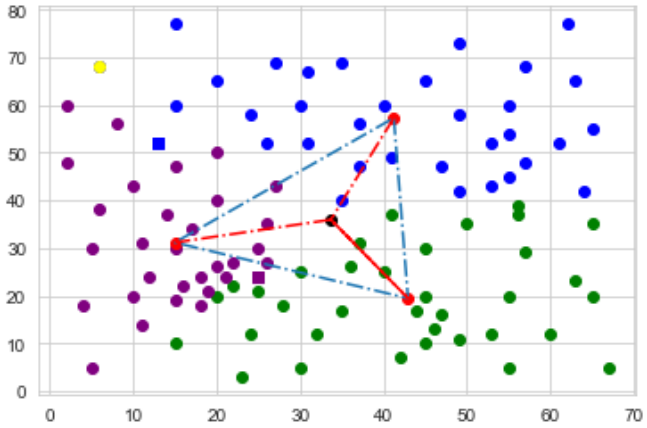


6-29Clustering step 2 with initial point i=49

Points in blue are those that have been assigned to the first salesman, those in green to the second and those to purple to the third. Points in red are the centers of gravity for all points and for each cluster separately and the point in yellow is the initial point i=64.

3. **Step 3: Problem of assigning customers to salesman**

The solution to the problem in step 3 showed that client 48 is moved from cluster 3 (purple) in step 2 to cluster 1 (blue) in step 3. Also, client 95 is moved from cluster 2 (green) in step 2 to cluster 3 (purple) in step 3. The next graph shows the new clustering based on turnover with the point that changed salesman being in a square.



6-30Clustering step3 with initial point i=49

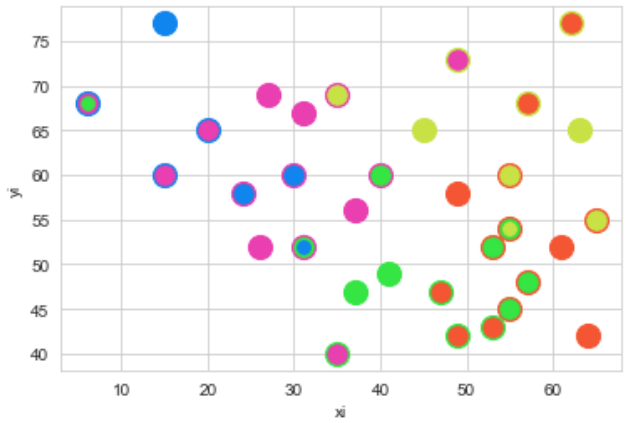
4. Step 4: Problem transferring customers from salesman to salesman

The solution to the problem in step 4 resulted in the optimal solution that points cannot return to its original clustering of step 2, i.e., the client's relocation request was not accepted. So in this case we chose to continue with the cluster of step 2

5. Step 5: Daily assignment of customers to salesmen

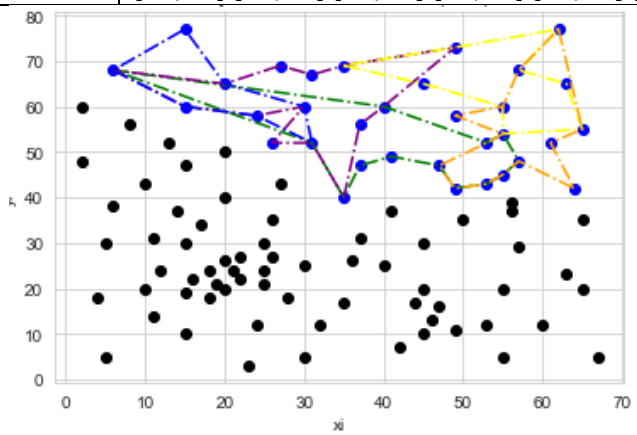
We first divide the assigned points into 5 clusters based on what has been mentioned in step 5. From the program we get the following results for each seller separately.

Salesman 1 (Blue):



6-31Daily Visits of salesman 1

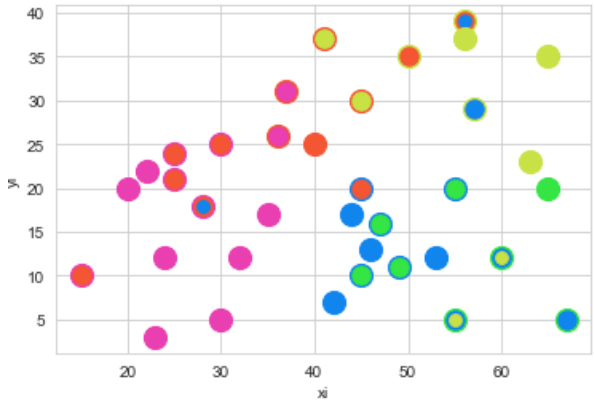
Days	Number of Points	Total Distance of the route for the day	Points of Route of the day for salesman 1 (Blue)
Green	13	135.695	[6,68] [40,60] [53,52] [57,48] [55,45] [53,43] [49,42] [47,47] [41,49] [37,47] [35,40] [31,52] [6,68]
Green	14	137	[6,68] [15,60] [24,58] [30,60] [26,52] [31,52] [35,40] [37,56] [40,60] [49,73] [35,69] [31,67] [27,69] [20,65] [6,68]
Yellow	9	93.363	[35,69] [45,65] [55,60] [55,54] [65,55] [63,65] [57,68] [62,77] [49,73] [35,69]
0Blue	7	75.452	[6,68] [15,60] [24,58] [31,52] [30,60] [20,5] [15,77] [6,68]
Orange	14	105.522	[62,77] [65,55] [61,52] [64,42] [57,48] [55,45] [53,43] [49,42] [47,47] [53,52] [55,54] [49,58] [55,60] [57,68] [62,77]



6-32Scatter Plot of salesman 1 visits in one period

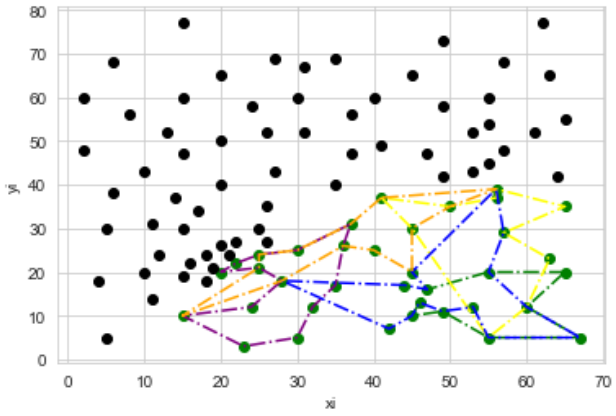
TOTAL DISTANCE OF SALESMAN 1 (BLUE): 547.032 distance units.

Salesman 2 (Green):



6-33Daily Visits of salesman 2

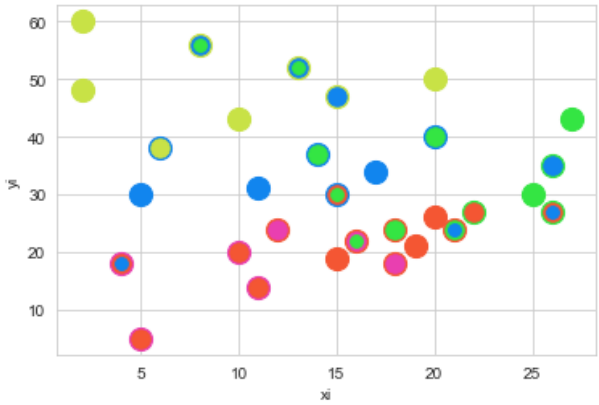
Days	Number of Points	Total Distance of the route for the day	Points of Route of the day for salesman 2(Green)
Green	8	69.21	[65,20] [55,20] [47,16] [45,10] [49,11] [55,5] [67,5] [0,12] [65,20]
Green	14	91.701	[37,31] [30,25] [25,24] [22,22] [20,20] [25,21] [28,18] [24,12] [15,10] [23,3] [30,5] [32,12] [35,17] [36,26] [37,31]
Yellow	10	100.872	[41,37] [45,30] [55,5] [60,12] [63,23] [57,29] [65,35] [56,39] [56,37] [50,35] [41,37]
0Blue	15	136.44	[56,39] [57,29] [55,20] [60,12] [67,5] [55,5] [53,12] [49,11] [46,13] [45,10] [42,7] [28,8] [44,17] [47,16] [45,20] [56,39]
Orange	13	116.583	[41,3] [37,31] [30,25] [25,24] [25,21] [15,10] [28,18] [36,26] [40,25] [45,20] [45,30] [50,35] [56,39] [41,37]



6-34Scatter Plot of salesman 2 visits in one period

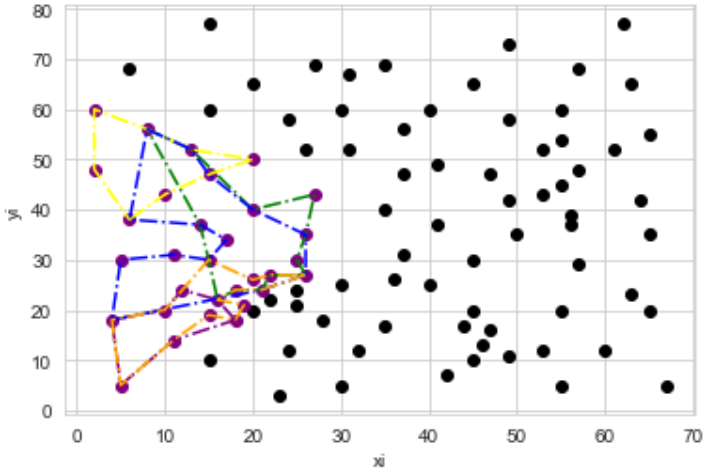
TOTAL DISTANCE OF SALESMAN 2 (GREEN): 514.806 distance units.

Salesman 3 (Purple):



6-35Daily Visits of salesman 3

Days	Number of Points	Total Distance of the route for the day	Points of Route of the day for salesman 3 (Purple)
Green	13	92.282	[26,27] [25,30] [26,35] [27,43] [20,40] [13,52] [8,56] [14,37] [15,30] [16,22] [18,24] [21,24] [22,27] [26,27]
Green	7	51.658	[18,18] [1,22] [12,24] [10,20] [4,18] [5,5] [11,14] [18,18]
Yellow	8	62.301	[6,38] [10,43] [15,47] [20,50] [13,52] [8,56] [2,60] [2,48] [6,38]
0Blue	14	117.195	[26,27] [21,24] [4,18] [5,30] [11,31] [15,30] [17,34] [14] [,37] [6,38] ,[8,56] [13,52] [15,47] [20,40] [26,35] [26,27],
Orange	15	81.547	[26,27] [21,24] [18,24] [16,22] [19,21] [18,18] [15,9] [11,14] [5,5] [4,18] [10,20] [12,24] [15,30] [20,26] [22,27] [26,27]

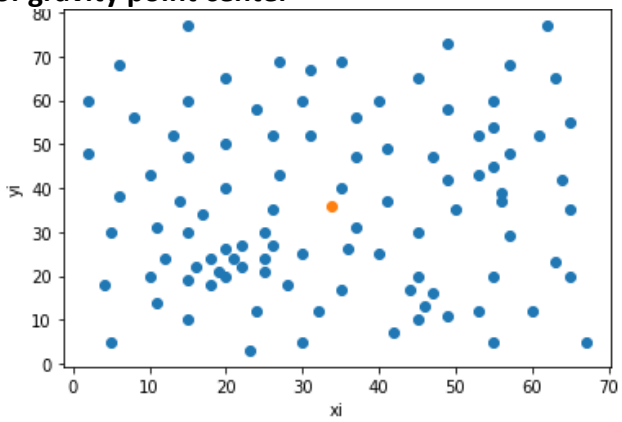


6-36Scatter Plot of salesman 3 visits in one period

TOTAL DISTANCE OF SALESMAN 3 (PURPLE): 404.983 distance units.

6.5 DAILY ROUTES AND DISTANCES FOR THE INITIAL POINT I= 38

1. **Step 1: Calculation of gravity point center**

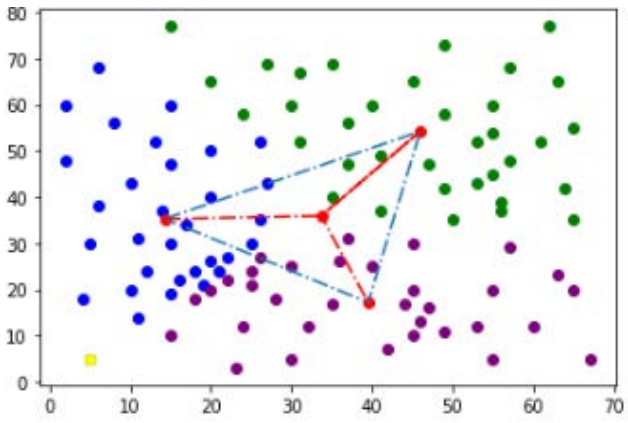


6-37Scatter Plot of Customers with gravity point center and initial point i=38

2. **Step 2: Ray scanning**

The furthest point from the center of gravity of the customer's points is i=38, $x_i = 5$, $y_i = 5$ with distance equal to 42.18692214419061 distance units.

Clustering based on geographical distribution, so every salesman would have 58 number of visits.

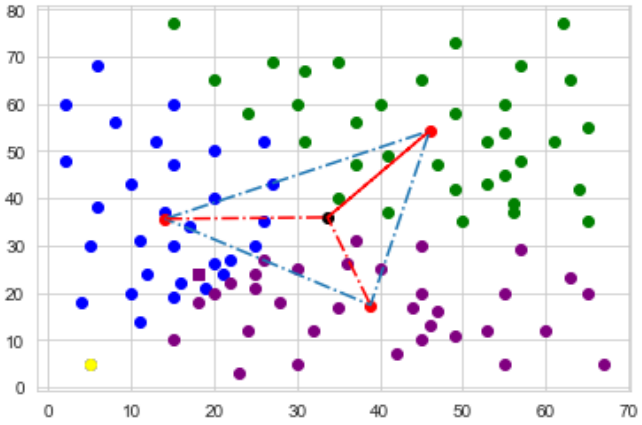


6-38Clustering step 2 with initial point i=38

Points in blue are those that have been assigned to the first salesman, those in green to the second and those to purple to the third. Points in red are the centers of gravity for all points and for each cluster separately and the point in yellow is the initial point i=38.

3. **Step 3: Problem of assigning customers to salesman**

The solution to the problem in step 3 showed that client 93 is moved from cluster 1 (blue) in step 2 to cluster 3 (purple) in step 3. The next graph shows the new clustering based on turnover with the point that changed salesman being in a square.



6-39Clustering step3 with initial point i=38

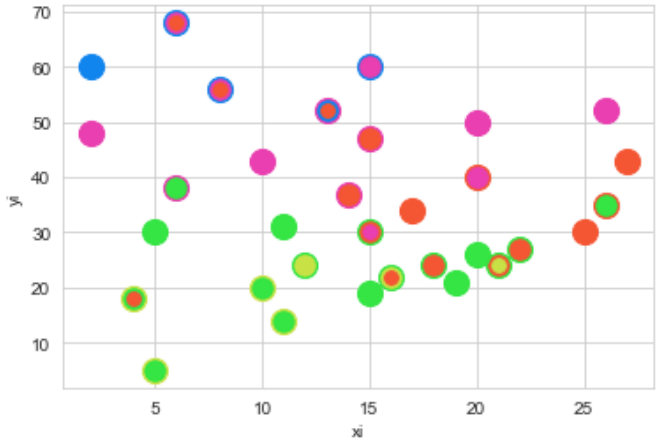
4. Step 4: Problem transferring customers from salesman to salesman

The solution to the problem in step 4 resulted in the optimal solution that points cannot return to its original clustering of step 2, i.e., the client's relocation request was not accepted. So in this case we chose to continue with the cluster of step 2

5. Step 5: Daily assignment of customers to salesmen

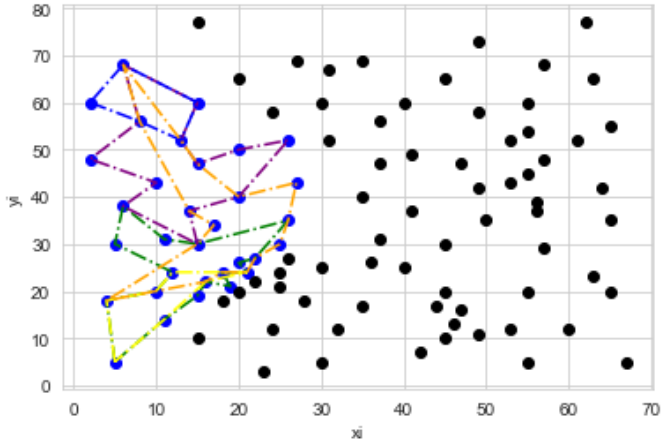
We first divide the assigned points into 5 clusters based on what has been mentioned in step 5. From the program we get the following results for each seller separately.

Salesman 1 (Blue):



6-40Daily Visits of salesman 1

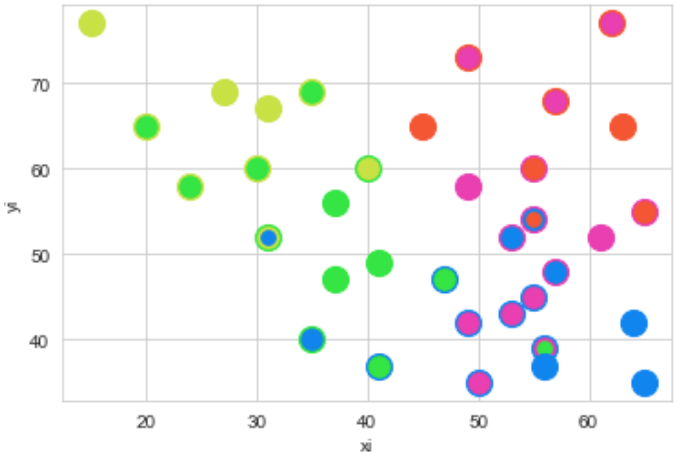
Days	Number of Points	Total Distance of the route for the day	Points of Route of the day for salesman 1 (Blue)
Green	17	109.047	[5,5] [11,14] [15,19] [16,22] [19,21] [18,24] [21,24] [20,26] [22,27] [26,35] [15,30] [11,31] [6,38] [5,30] [12,24] [10,20] [4,18] [5,5]
Green	13	115.069	[15,30] [14,37] [20,40] [26,52] [20,50] [15,47] [13,52] [15,60] [6,68] [8,56] [2,48] [10,43] [6,38] [15,30]
Yellow	7	58.471	[5,5] [4,18] [10,20] [12,24] [21,24] [16,22] [11,14] [5,5]
0Blue	5	42.846	[2,60] [8,56] [13,52] [15,60] [6,68] [2,60]
Orange	16	135.195	[21,24] [22,27] [25,30] [26,35] [37,43] [20,40] [15,47] [13,52] [6,68] [8,56] [14,37] [17,34] [15,30] [4,18] [16,22] [18,24] [21,24]



6-41Scatter Plot of salesman 1 visits in one period

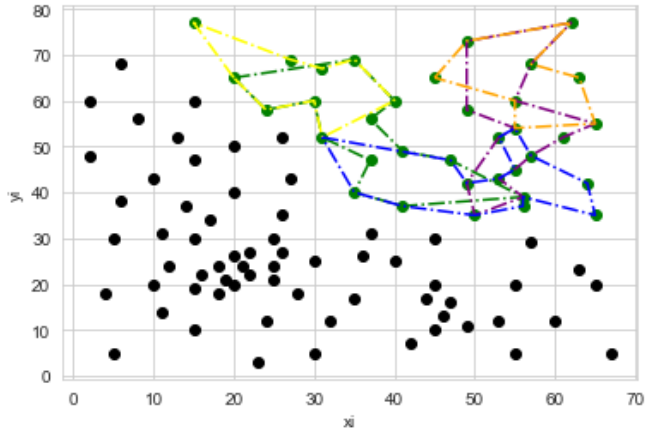
TOTAL DISTANCE OF SALESMAN 1 (BLUE): 460.628 distance units.

Salesman 2 (Green):



6-42Daily Visits of salesman 2

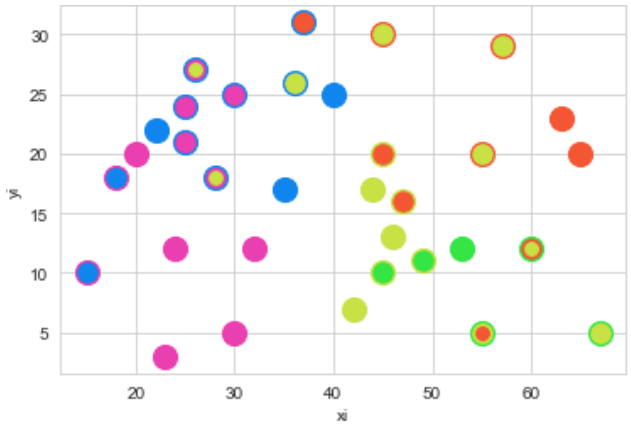
Days	Number of Points	Total Distance of the route for the day	Points of Route of the day for salesman 2(Green)
Green	13	116.629	[20,65] [35,69] [40,60] [37,56] [41,49] [47,47] [56,39] [41,37] [35,40] [37,47] [31,52] [30,6] [24,58] [20,65]
Green	15	115.505	[49,73] [49,58] [55,54] [53,52] [49,42] [50,35] [56,39] [53,43] [55,45] [57,48] [61,52] [65,55] [55,60] [57,68] [62,77] [49,73]
Yellow	9	81.153	[20,65] [15,77] [27,69] [31,67][35,69] [40,60] [31,52] [30,60] [24,58] [20,65]
0Blue	15	108.574	[31,52] [47,47] [49,42] [53,43] [55,45] [53,52] [55,54] [57,48] [64,42] [65,35] [56,39] [56,37] [50,35] [41,37] [35,40] [31,52]
Orange	8	76.977	[45,65] [49,73] [62,77] [57,68] [63,65] [65,55] [55,54] [55,60] [45,65]



6-43Scatter Plot of salesman 2 visits in one period

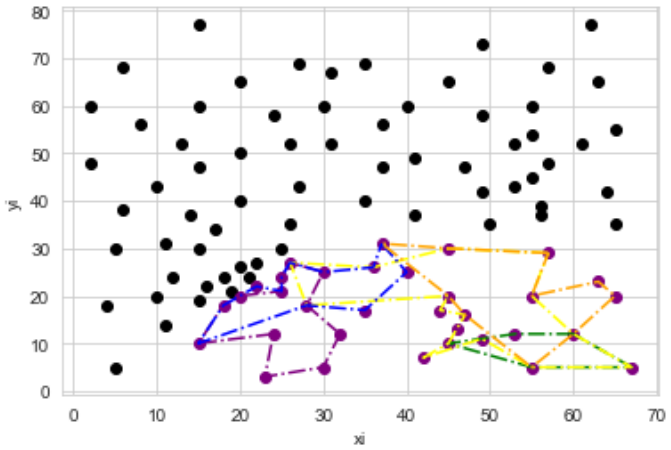
TOTAL DISTANCE OF SALESMAN 2 (GREEN): 498.835 distance units.

Salesman 3 (Purple):



6-44Daily Visits of salesman 3

Days	Number of Points	Total Distance of the route for the day	Points of Route of the day for salesman 3 (Purple)
Green	6	48.325	[60,12] [53,12] [49,11] [45,10] [5,5] [67,5] [60,12]
Green	12	74.431	[32,12] [28,18] [30,25] [36,7] [25,24] [25,21] [20,20] [18,18] [15,10] [24,12] [23,3] [30,5] [32,12]
Yellow	16	132.269	[57,29] [45,30] [36,26] [26,27] [28,18] [45,20] [44,17] [47,16] [46,13] [45,10] [42,7] [49,11] [55,5] [67,5] [60,12] [55,20] [57,29]
0Blue	12	77.656	[37,31] [40,25] [35,17] [28,18] [15,10] [18,18] [22,22] [25,21] [25,24] [26,27] [30,25] [36,26] [37,31]
Orange	10	91.184	[57,29] [45,30] [37,31] [45,20] [47,16] [55,5] [60,12] [65,20] [63,23] [55,20] [57,29]

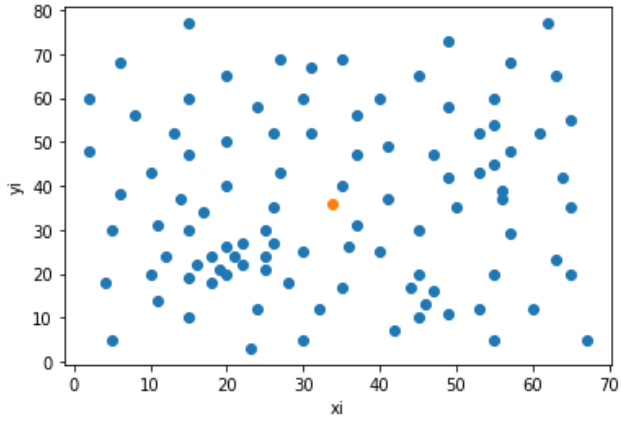


6-45Scatter Plot of salesman 3 visits in one period

TOTAL DISTANCE OF SALESMAN 3 (PURPLE): 423.865 distance units.

6.6 DAILY ROUTES AND DISTANCES FOR THE INITIAL POINT I= 35

1. Step 1: Calculation of gravity point center

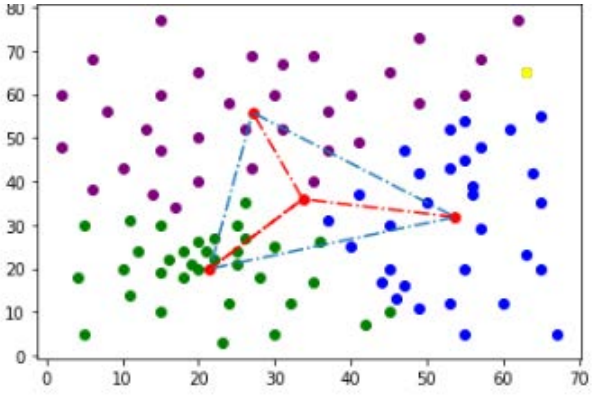


6-46Scatter Plot of Customers with gravity point center and initial point i=35

2. Step 2: Ray scanning

The furthest point from the center of gravity of the customer's points is i=35, $x_i = 63$, $y_i = 65$ with distance equal to 41.28118699843792 distance units.

Clustering based on geographical distribution, so every salesman would have 58 number of visits.

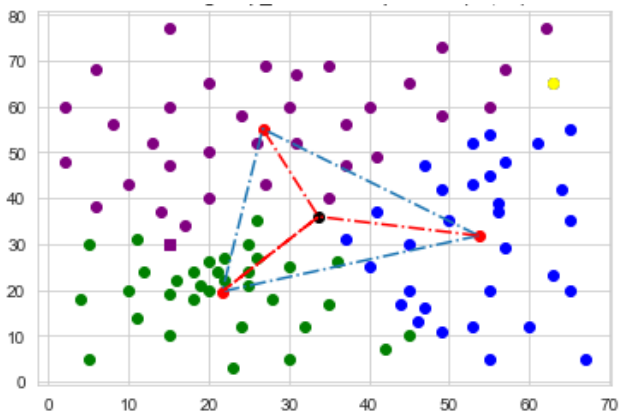


6-47Clustering step 2 with initial point i=35

Points in blue are those that have been assigned to the first salesman, those in green to the second and those to purple to the third. Points in red are the centers of gravity for all points and for each cluster separately and the point in yellow is the initial point i=35.

3. Step 3: Problem of assigning customers to salesman

The solution to the problem in step 3 showed that client 5 is moved from cluster 2 (green) in step 2 to cluster 3 (purple) in step 3. The next graph shows the new clustering based on turnover with the point that changed salesman being in a square.



6-48Clustering step3 with initial point i=35

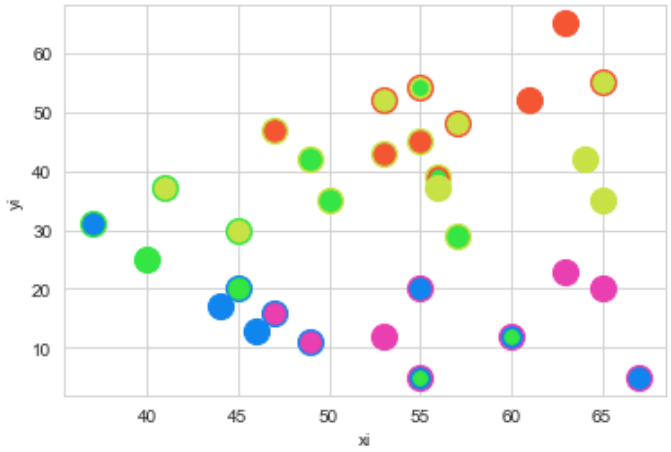
4. Step 4: Problem transferring customers from salesman to salesman

The solution to the problem in step 4 resulted in the optimal solution that points cannot return to its original clustering of step 2, i.e., the client's relocation request was not accepted. So in this case we chose to continue with the cluster of step 2

5. Step 5: Daily assignment of customers to salesmen

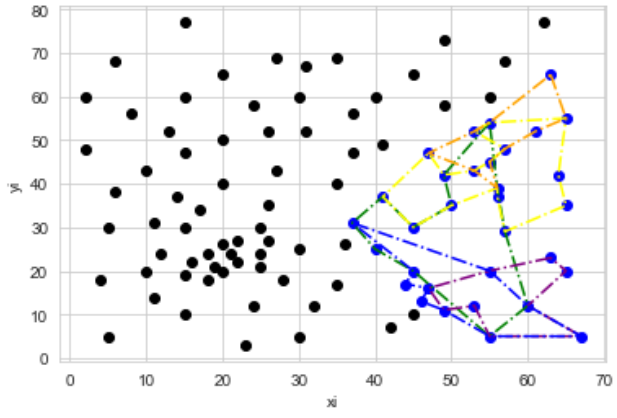
We first divide the assigned points into 5 clusters based on what has been mentioned in step 5. From the program we get the following results for each seller separately.

Salesman 1 (Blue):



6-49Daily Visits of salesman 1

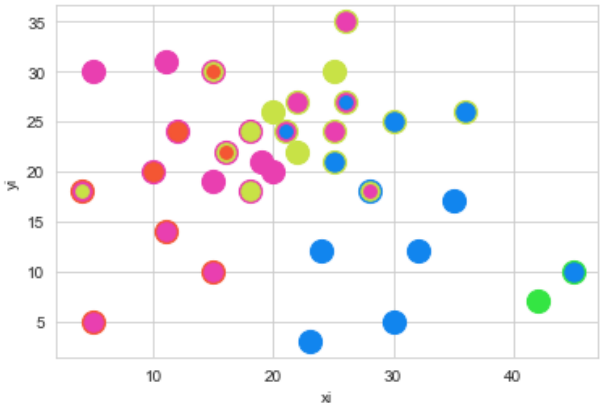
Days	Number of Points	Total Distance of the route for the day	Points of Route of the day for salesman 1 (Blue)
Green	12	125.586	[55,54] [49,42] [50,35] [45,30] [41,37] [37,31] [40,25] [45,20] [55,5] [60,12] [57,29] [56,39] [55,54]
Green	9	69.215	[63,23] [55,20] [47,16] [49,11] [53,12] [55,5] [67,5] [60,12] [65,20] [63,23]
Yellow	16	108.654	[55,54] [53,52] [57,48] [55,45] [53,43] [49,42] [47,47] [41,37] [45,30] [50,35] [56,39] [56,37] [57,29] [65,35] [64,42] [65,55] [55,54]
0Blue	10	87.606	[55,20] [660,12] [67,5] [55,5] [49,11] [46,13] [47,16] [44,17] [45,20] [37,31] [55,20]
Orange	10	66.994	[63,65] [65,55] [61,52] [57,48] [55,45][56,39] [53,43] [47,47] [53,52] [55,54] [63,65]



6-50Scatter Plot of salesman 1 visits in one period

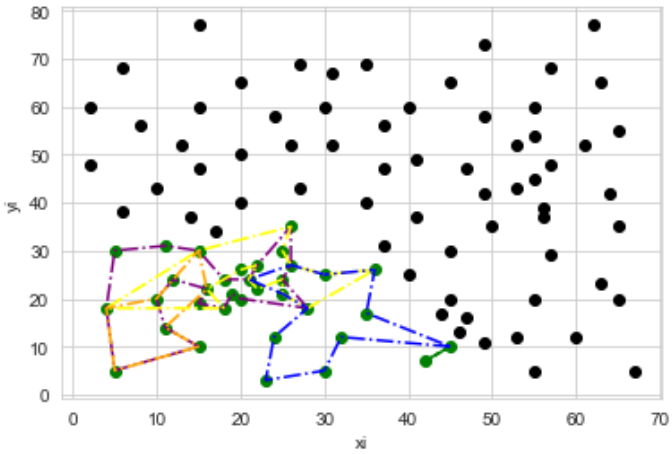
TOTAL DISTANCE OF SALESMAN 1 (BLUE): 458.055 distance units.

Salesman 2 (Green):



6-51Daily Visits of salesman 2

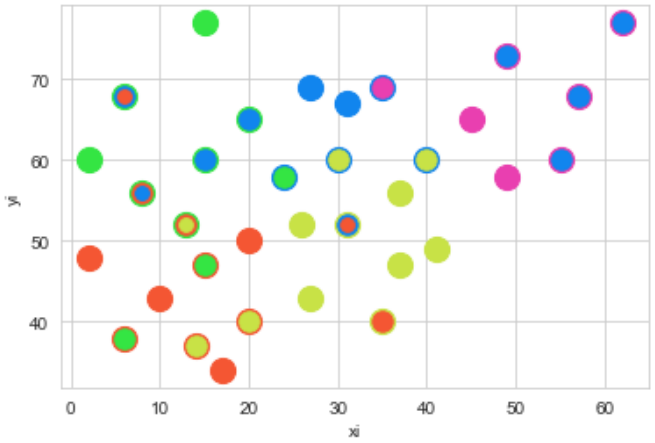
Days	Number of Points	Total Distance of the route for the day	Points of Route of the day for salesman 2(Green)
Green	2	8.486	[45,10] [42,7] [45,10]
Green	21	125.98	[28,18] [25,24] [26,27] [26,35] [22,27] [21,24] [18,24] [15,30] [11,31] [5,30] [4,18] [5,5] [15,10] [11,14] [10,20] [12,24] [16,22] [15,19] [18,18] [19,21] [20,20] [28,18]
Yellow	17	101.103	[36,26] [30,25] [26,27] [25,30] [26,35] [15,30] [4,18] [18,18] [16,22] [18,24] [20,26] [22,27] [21,24] [22,22] [25,24] [25,21] [28,18] [36,26]
Blue	12	98.87	[45,10] [32,12] [30,5] [23,3] [24,12] [28,18] [25,21] [21,24] [26,27] [30,25] [36,26] [35,17] [45,10]
Orange	8	64.876	[15,10] [5,5] [4,18] [10,20] [12,24] [15,30] [16,22] [11,14] [15,10]



6-52Scatter Plot of salesman 2 visits in one period

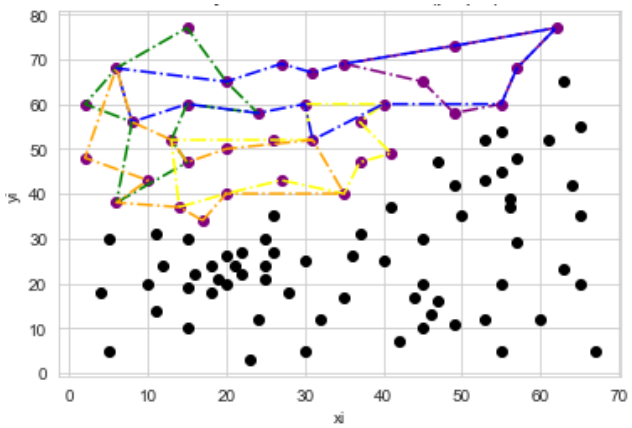
TOTAL DISTANCE OF SALESMAN 2 (GREEN): 399.315 distance units.

Salesman 3 (Purple):



6-53Daily Visits of salesman 3

Days	Number of Points	Total Distance of the route for the day	Points of Route of the day for salesman 3(Purple)
Green	10	103.635	[6,38] [15,47] [13,52] [15,60] [24,58] [20,65] [15,77] [6,68] [2,60] [8,56] [6,38]
Green	7	71.86	[35,69] [45,65] [49,58] [55,60] [57,68] [62,77] [49,73] [35,69]
Yellow	12	98.777	[14,37] [20,40] [7,43] [35,40] [37,47] [41,49] [37,56] [40,60] [30,60] [31,52] [26,52] [13,52] [14,37]
Blue	15	148.904	[8,56] [6,68] [20,65] [27,69] [31,67] [35,69] [49,73] [62,77] [57,68] [55,60] [40,60] [31,52] [30,60] [24,58] [15,60] [8,56]
Orange	13	123.86	[17,34] [14,37] [6,38] [10,43] [2,48] [6,68] [8,56] [13,52] [15,47] [20,50][31,52] [35,40] [20,40] [17,34]

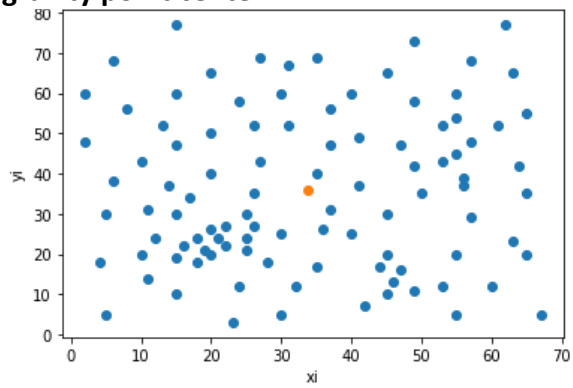


6-54Scatter Plot of salesman 3 visits in one period

TOTAL DISTANCE OF SALESMAN 3 (PURPLE): 547.036 distance units.

6.7 DAILY ROUTES AND DISTANCES FOR THE INITIAL POINT I= 66

1. **Step 1: Calculation of gravity point center**

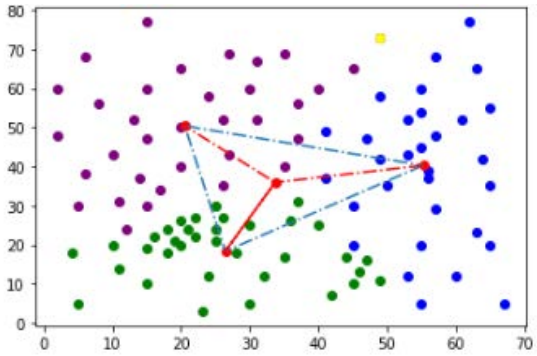


6-55Scatter Plot of Customers with gravity point center and initial point i=66

2. **Step 2: Ray scanning**

The furthest point from the center of gravity of the customer's points is i=66, $x_i = 49$, $y_i = 73$ with distance equal to 40.112546665600775 distance units.

Clustering based on geographical distribution, so every salesman would have 58 number of visits.

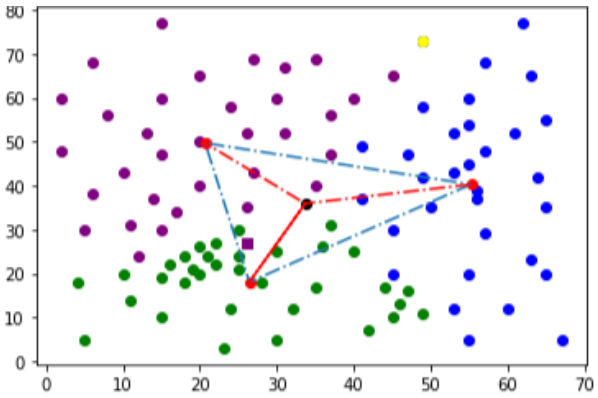


6-56Clustering step 2 with initial point i=66

Points in blue are those that have been assigned to the first salesman, those in green to the second and those to purple to the third. Points in red are the centers of gravity for all points and for each cluster separately and the point in yellow is the initial point i=66.

3. **Step 3: Problem of assigning customers to salesman**

The solution to the problem in step showed that client 94 is moved from cluster 2 (green) in step 2 to cluster 3 (purple) in step 3. The next graph shows the new clustering based on turnover with the point that changed salesman being in a square.



6-57Clustering step3 with initial point i=66

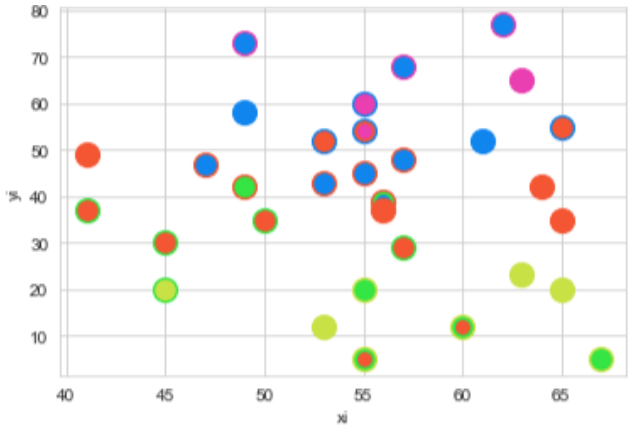
4. **Step 4: Problem transferring customers from salesman to salesman**

The solution to the problem in step 4 resulted in the optimal solution that points cannot return to its original clustering of step 2, i.e., the client's relocation request was not accepted. So in this case we chose to continue with the cluster of step 2

5. **Step 5: Daily assignment of customers to salesmen**

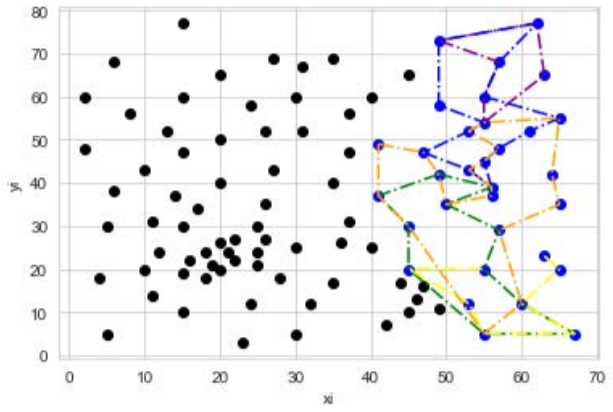
We first divide the assigned points into 5 clusters based on what has been mentioned in step 5. From the program we get the following results for each seller separately.

Salesman 1 (Blue):



6-58Daily Visits of salesman 1

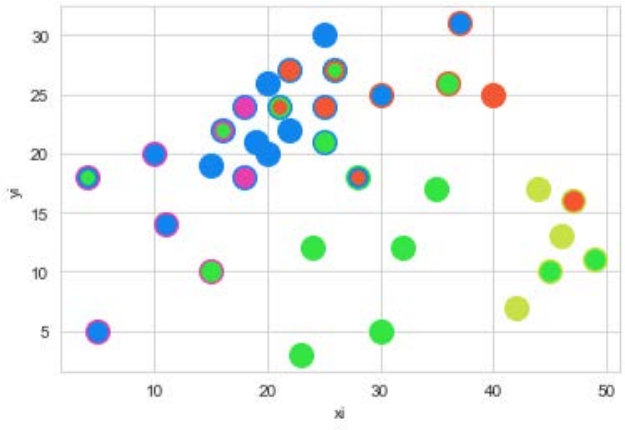
Days	Number of Points	Total Distance of the route for the day	Points of Route of the day for salesman 1 (Blue)
Green	11	110.124	[49,42] [41,37] [45,30] [45,20] [55,5] [67,5] [60,12] [55,20] [57,29] [50,35] [56,39] [49,42]
0,35Green	6	62.924	[49,73] [57,68] [5560] [55,54] [63,65] [62,77] [49,73]
Yellow	8	72.077	[63,23] [65,20] [60,12] [67,5] [55,5] [53,12] [45,20] [55,20]
Blue	14	108.729	[49,73] [49,58] [55,54] [53,52] [47,47] [53,43] [46,39] [55,45] [57,48] [61,52] [65,55] [55,60] [57,68] [62,77] [49,73]
Orange	19	160.037	[41,49] [47,47] [49,42] [50,35] [56,37] [56,39] [53,43] [55,45] [57,48] [53,52] [55,54] [65,55] [64,42] [65,35] [57,29] [60,12] [55,5] [45,30] [41,37] [41,49]



6-59Scatter Plot of salesman 1 visits in one period

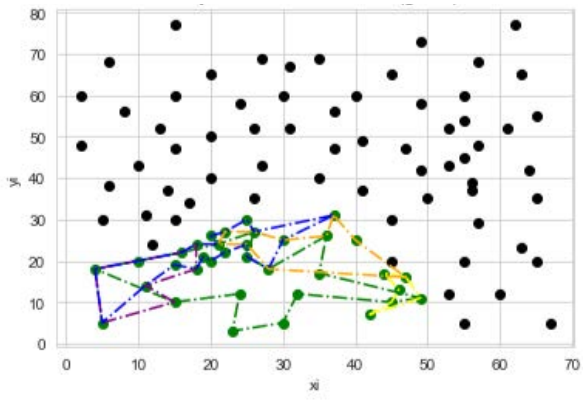
TOTAL DISTANCE OF SALESMAN 1 (BLUE): 513.891 distance units.

Salesman 2 (Green):



6-60Daily Visits of salesman 2

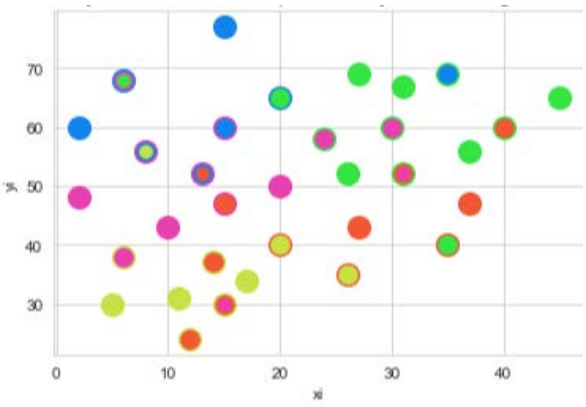
Days	Number of Points	Total Distance of the route for the day	Points of Route of the day for salesman 2(Green)
Green	15	133.504	[49,11][45,10] [32,12] [30,] [23,3] [24,12] [15,10] [4,18] [16,22][21,24] [26,27] [25,21] [28,18] [36,26] [35,17] [49,11]
Green	8	59,415	[15,10] [11,14] [18,18] [18,24] [16,22] [10,20] [4,18] [5,5] [15,10]
Yellow	6	28,486	[47,16] [44,17] [46,13] [45,10] [42,7] [49,11] [47,16]
Blue	21	110.233	[37,31] [26,27] [25,30] [22,27] [20,26] [21,24] [18,24] [16,22] [10,20] [4,18] [5,5] [11,14] [15,19] [18,18] [19,21] [20,20] [22,22] [25,24] [25,21] [28,18] [30,25][37,31]
Orange	10	70.739	[37,31] [36,26] [30,25] [26,27] [22,27] [21,24] [25,24] [28,18] [47,16] [40,25] [37,31]



6-61Scatter Plot of salesman 2 visits in one period

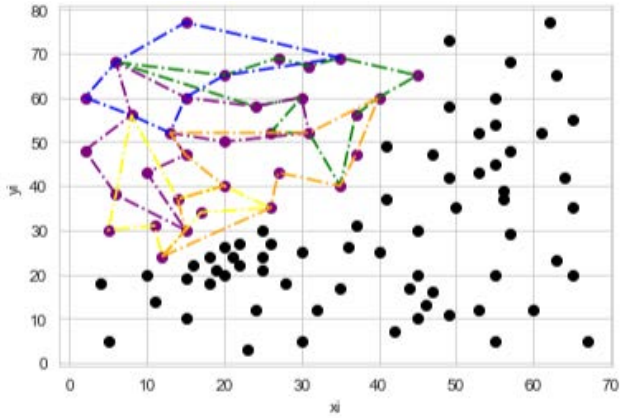
TOTAL DISTANCE OF SALESMAN 2 (GREEN): 402.377 distance units.

Salesman 3 (Purple):



6-62Daily Visits of salesman 3

Days	Number of Points	Total Distance of the route for the day	Points of Route of the day for salesman 3(Purple)
Green	13	123.799	[6,68] [20,65] [27,69] [31,67] [35,69] [45,65][40,60] [37,56] [35,40] [31,52] [26,52] [30,60] [24,58] [6,68]
0,35Green	13	124.803	[15,30] [6,38] [2,48] [8,56] [6,68] [15,60] [24,58] [30,60] [31,52] [20,50] [13,52] [15,47] [10,43] [15,30]
Yellow	10	94.005	[12,24] [15,30] [17,34] [26,35] [20,4] [14,37] [8,56] [6,38] [5,30] [11,31] [12,24]
Blue	8	87.668	[2,60] [8,56] [13,52] [15,60] [20,65] [35,69] [15,77] [6,68] [2,60]
Orange	12	119.548	[12,24] [26,35] [27,43] [35,40] [37,47] [40,60] [31,52] [13,52] [15,47] [20,40] [14,37] [15,30] [12,24]



6-63Scatter Plot of salesman 3 visits in one period

TOTAL DISTANCE OF SALESMAN 3 (PURPLE): 549.868 distance units.

7 CONCLUSIONS

The study of how to formally transform routing problems such as the TSP, VRP, and PVRP into mathematical models has profound real-world implications across a wide range of industries. Effective mathematical modeling enables companies and public institutions to make data-driven decisions that significantly improve operational efficiency, reduce costs, and enhance service quality. For instance, consider a grocery distribution company that delivers products to hundreds of retail stores across a region. Some stores require daily replenishment, while others may only need service two or three times per week. By transforming this logistical scenario into a Periodic Vehicle Routing Problem (PVRP) model, the company can determine optimal delivery schedules for each store, assign routes to vehicles efficiently, and minimize total travel distance and labor costs over the planning horizon. Without such a model, planners would have to rely on intuition or trial-and-error, often leading to redundant trips, underutilized vehicles, and higher operational expenses. Moreover, in urban services like postal delivery or waste collection, structured PVRP models allow for consistent, cost-effective service delivery while adapting to city-specific constraints like traffic patterns and service time windows. As sustainability and environmental concerns grow, these models also support emission reduction by optimizing routes and limiting unnecessary mileage. Ultimately, studying the transformation of real-world routing challenges into mathematical formulations enables organizations to bridge the gap between theoretical optimization and practical impact, making operations more efficient, adaptive, and sustainable.

In this paper, an algorithm was developed to solve a simple form of the PVRP problem, where 100 customers/points should be assigned fairly to 3 salesmen based on both their geographical distribution in space and the turnover disbursement from each customer/point.

Following the 5 steps of the algorithm, the following distances that each salesman should travel in one period were derived. A small statistical analysis was performed on this data by calculating the average value and standard deviation of these values. According to the results, the best solution is obtained by applying the algorithm with the customer as the starting point i=38

Initial Point	BLUE	GREEN	PURPLE	Average Value	Standard Deviation
65	485,66	374,84	550,11	470,20	72,38
67	444,72	499,76	528,39	490,95	34,72
64	498,59	471,50	420,33	463,47	32,45
49	547,03	514,81	404,98	488,94	60,81
38	460,63	498,84	423,87	461,11	30,61
35	458,06	399,32	547,04	468,14	60,73
66	513,89	402,38	549,87	488,71	62,79

Table 3 Aggregated table of cumulative distances for each seller in a period

Although the algorithm is built to produce the best possible balance between geographical distribution and turnover disbursement by sellers, there are still steps to improve it.

For example, in the fifth step where the points have already been allocated to the vendors, virtual points with the same coordinates have been created in order to simplify the concept of periodicity and essentially have points that require only one visit within the period and the k-means algorithm is applied to distribute the points of a period into five sub-regions, each of which represents one day of the week, an optimization problem could be applied so that the sub-areas containing a few points relative to the rest are filtered out of the larger ones in order to avoid the situation where a vendor visits 21 points one day and 2 points the next. This could be done by setting a bound for when a sub-region is considered small compared to the others. This calculation may need to consider the number of points entering each sub-region and the distance the seller will need to travel to these points.

8 REFERENCES

[1] G.B.Dantzigand J.H.Ramer, “The Truck Dispatching Problem”, Management Sci 6, 80-91 (1959) and “Optimum Routing of Gasoline Delivery Trucks” Proc. Fifth World Petroleum Cong, ection VIII, p.. 19 (1959).

[2] G. Clarke and J.W.Wright “Scheduling of vehicles from a central depot to a number of delivery points”

[3] Beltrami, E. J. and L. D. Bodin (1974). Networks and vehicle routing for municipal waste collection. Networks 4(1), 65–94.

[4] Russell, R. A. and W. Igo (1979). An assignment routing problem. Networks 9 (1), 1–17.

[5] Christofides, N. and J. E. Beasley (1984). The period routing problem. Networks 14 (2), 237–256.

[6] Golden, B. L., T. L. Magnanti, and H. Q. Nguyen (1977). Implementing vehicle routing algorithms. Networks 7(2), 113–148.

[7] Tan, C. C. R. and J. E. Beasley (1984). A heuristic algorithm for the period vehicle routing problem. Omega 12(5), 497–504.

[8] Traveling Salesman Problem (TSP) with Miller-Tucker-Zemlin (MTZ) in CPLEX/OPL – Coenzyme (co-enzyme.fr)

[9] Hadjiconstantinou, E. and R. Baldacci (1998). A multi-depot period vehicle routing problem arising in the utilities sector. The Journal of the Operational Research Society 49 (12), 1239–1248.

[10] Cordeau, J.-F., G. Laporte, and A. Mercier (2001). A unified tabu search heuristic for vehicle routing problems with time windows. Journal of the Operational Research Society 52 (8), 928–936.

[11] Francis, P. M., and K. R. Smilowitz (2006). Modeling techniques for periodic vehicle routing problems. Transportation Research: Part B 40 (10), 872–884.

[12] D. Gulczynski, J. Heath, C. Price. The Close Enough Traveling Salesman Problem: A Discussion of Several Heuristics. Perspectives in Operations Research: Papers in Honor of Saul Gass’ 80th Birthday, F. Alt, M. Fu, and B. Golden,271–283, Springer Verlag, 2006.

[13] A. Imai, E. Nishimura, and S. Papadimitriou. The dynamic berth allocation problem for a container port. *Transportation Research-B*, 35:401–417, 2001.

9 APPENDIX

9.1 N. CRISTOFIDES AND J.E.BEASSLEY PAPERS DATA SET FOR 100D PROBLEM

https://people.brunel.ac.uk/~mastjjb/jeb/orlib/periodinfo.html

From the above library we can download the data in text document format which means that we have to edit it before we can use it in our code. For this purpose, we used Python, one of the best choices in the latest generation of programming languages with a lot of data science capabilities.

```
import re
import pandas as pd
# Open these two files with read/write permissions
file1 = open ("data_100d.txt", "r+")
file2 = open ("data_100d_mod.txt", "r+")
file3 = open ("data_100d_final.txt", "r+")
for x in range (1, 101):
 # Remove the leading and trailing whitespaces on each line of data_100d.txt and write the formatted lines in data_100d_mod.txt
   line = file1.readline()
   line = line.lstrip()
   line = line.rstrip()
   file2.write(line + "\n")
file2.close()
file2 = open ("data_100d_mod.txt", "r+")
# Modify the data_100d_mod.txt file in a comma-separated format as to create .csv file
for x in range (1, 101):
   line = file2.readline()
   line = re.sub("\s", ",", line)
   file3.write(line + "\n")
dataframe1 = pd.read_csv("data_100d_final.txt")
```

dataframe1.to_csv ("data_100d_final.csv", index = None)

file1.close()

file2.close()

file3.close()

In our problem, we have 100 number of customers, length of period 5 days, 3 salesmen and the frequency of visits depends on the demand of every customer in accordance to the following constraints:

- If the demand of the customer is more or equal to 10, then the customer requires one delivery in period.
- If the demand of the customer is more or equal to 11 and less or equal to 25, then the customer requires 2 deliveries in a period.
- If the demand of the customer is more or equal to 26, then the customer requires 3 deliveries in period.

So, we have the following data table for the customers (x_i, y_i), the load q_i they require and the frequency $u_i$with which they require it. At this point we assume that q_i is requested at each visit, so customer 3 requires a total of 26 quantities of the product or service that the seller has undertaken.

i	xi	yi	qi
1	41	49	10
2	35	17	7
3	55	45	13
4	55	20	19
5	15	30	26
6	25	30	3
7	20	50	5
8	10	43	9
9	55	60	16
10	30	60	16
11	20	65	12
12	50	35	19
13	30	25	23
14	15	10	20
15	30	5	8
16	10	20	19
17	5	30	2
18	20	40	12
19	15	60	17
20	45	65	9
21	45	20	11
22	45	10	18

23	55	5	29
24	65	35	3
25	65	20	6
26	45	30	17
27	35	40	16
28	41	37	16
29	64	42	9
30	40	60	21
31	31	52	27
32	35	69	23
33	53	52	11
34	65	55	14
35	63	65	8
36	2	60	5
37	20	20	8
38	5	5	16
39	60	12	31
40	40	25	9
41	42	7	5
42	24	12	5
43	23	3	7
44	11	14	18
45	6	38	16
46	2	48	1
47	8	56	27
48	13	52	36
49	6	68	30
50	47	47	13
51	49	58	10
52	27	43	9
53	37	31	14
54	57	29	18
55	63	23	2
56	53	12	6
57	32	12	7
58	36	26	18
59	21	24	28
60	17	34	3
61	12	24	13
62	24	58	19
63	27	69	10
64	15	77	9
65	62	77	20
66	49	73	25

67	67	5	25
68	56	39	36
69	37	47	6
70	37	56	5
71	57	68	15
72	47	16	25
73	44	17	9
74	46	13	8
75	49	11	18
76	49	42	13
77	53	43	14
78	61	52	3
79	57	48	23
80	56	37	6
81	55	54	26
82	15	47	16
83	14	37	11
84	11	31	7
85	16	22	41
86	4	18	35
87	28	18	26
88	26	52	9
89	26	35	15
90	31	67	3
91	15	19	1
92	22	22	2
93	18	24	22
94	26	27	27
95	25	24	20
96	22	27	11
97	25	21	12
98	19	21	10
99	20	26	9
100	18	18	17

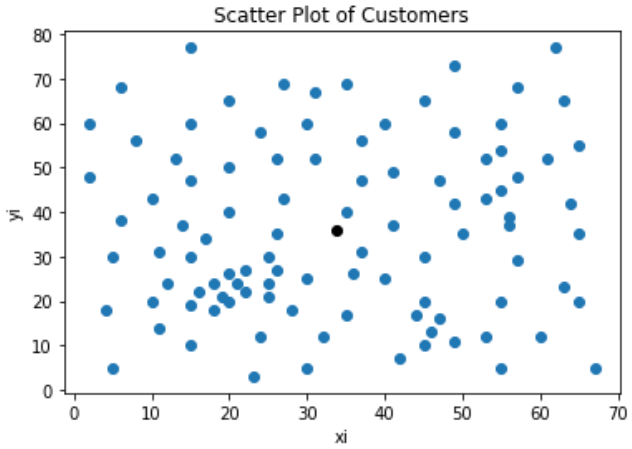
9.2 PYTHON CODE FOR THE ALGORITHM OF CHAPTER 5

Step 1: Calculation of gravity point center

As discussed above in chapter 5, the first step is to calculate the center of gravity of the customer points. This can be done with the following code.

```
#These libraries are required for data processing and graphical representation

#Read the data from the file we have saved

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import math

data = pd.read_csv("C:/Users/kater/Desktop/PVRP/PRESENTATIONS/data_100d_final.csv")

data.index = np.arange(1, len(data)+1)

plt.scatter(data['xi'], data['yi'])

plt.title("Scatter Plot of Customers")

plt.xlabel('xi')

plt.ylabel('yi')

#Customer center of Gravity Calculation

xmean=0

ymean=0

N=len(data)

xmean=data['xi'].sum()/N

ymean=data['yi'].sum()/N

print("The customers' centre of gravity is: ", "xmean= ",xmean,'ymean= ',ymean)

 #Display in a diagram

plt.scatter(xmean,ymean,c="black")

plt.show()
```

Up to this point we get the result shown in the following cutout with the center of gravity at the coordinates $x_{mean} = 33{,}7$ and $y_{mean} = 35{,}92$.



Step 2: Ray scanning

In this step we divide the customer points into 3 regions so that each region assigned to each salesman contains points with a total number of visits equal to the average number of visits. For this purpose we will need to calculate some quantities and some distances explained below.

We first calculate the **number of visits u_i** for each client based on the assumptions made by the article's conveners as described in Appendix 1:

```
ui=[]
for i in range(1, N+1):
    if data['qi'][i]<= 10:
        ui.append(1)
    elif data['qi'][i]>=11 and data['qi'][i]<=25:
        ui.append(2)
    else:
        ui.append(3)
    #print('x'+str(i),':',data['xi'][i],'/','y'+str(i),':',data['yi'][i],'/','q'+str(i),':',data['qi'][i],'/','u'+str(i),':',ui[i-1])
print('\n')
```

We could place the data in an excel to process them in this format if we want
data.to_excel('data_set_frequency.xlsx')

We calculate the distance of points from the center of gravity point of the customer cloud

```
di=[[]*2]*N
#di.insert(1,[1,1])
for i in range(1,N+1):
    di.insert(i-1,[i,math.sqrt((xmean-data['xi'][i])**2+(ymean-data['yi'][i])**2)])
    #print(i,data['xi'][i])
    di.pop() # removes the last item from the list on each iteration
```

We classify the customers based on distance from the center point of the cloud. Specifically, we ask to list the points in order from farthest to closest to the center of gravity.

```
di_sorted=sorted(di,key=lambda l:l[1],reverse=True)
#print('Classification of customers based on distance from the center of gravity')
print(di_sorted)
#print('\n')
```

We calculate the distance of points from the (0,0) of the Cartesian plane.

```
di_zero=[[]*2]*N
#di.insert(1,[1,1])
for i in range(1,N+1):
    di_zero.insert(i-1,[i,math.sqrt((0-data['xi'][i])**2+(0-data['yi'][i])**2)])
    #print(i,data['xi'][i])
    di_zero.pop() # removes the last item from the list on each iteration
#print('Distances from the beginning of the axes')
#print(di_zero,'\n')
```

We calculate the average number of visits for each salesman

```
 data.insert(4,'ui',ui,True)
data.drop(data.columns[data.columns.str.contains('unnamed', case=False)], axis=1, inplace=True)
umean=0
umean=data['ui'].sum()/(saleman)
print(' The average number of visits corresponding to each salesperson is equal to ',umean,'\n')
```

We calculate the average order size

```
saleman=3
qmean=0
qmean=(data['qi']*data['ui']).sum()/saleman
print('  The average order size is equal to ',qmean,'\n')
```

We create the clusters

```
print(' As a starting point we choose (62,77) corresponding to point 65 ','\n')
cluster=saleman
initial_x=data['xi'][di_sorted[2][0]]
initial_y=data['yi'][di_sorted[2][0]]
```

We calculate the angle of rotation of the coordinate system

```
rotate_angle=math.acos((initial_x-xmean)/di[di_sorted[2][0]-1][1])
rotate_angle_degrees=180*rotate_angle/math.pi
print('Γωνία περιστροφής σε ακτίνια:',rotate_angle,'και σε μοίρες:',rotate_angle_degrees, '\n')
#print(di[di_sorted[0][0]][1])
```

We rotate our system based on the above angle

```
data_rotated=data.copy()
data_rotated['xi'] = data_rotated['xi'].astype(float)
data_rotated['yi'] = data_rotated['yi'].astype(float)
for i in range(1,N+1):
   x_rot=data['xi'][i]-xmean
   y_rot=data['yi'][i]-ymean
   data_rotated['xi'][i]=x_rot*math.cos(rotate_angle)+y_rot*math.sin(rotate_angle)
   data_rotated['yi'][i]=y_rot*math.cos(rotate_angle)-x_rot*math.sin(rotate_angle)
#print( ' The new points of the system that has been circumvented are the following: ')
#print(data_rotated)
data_rotated.to_excel('data_rotated.xlsx')
```

Calculate the angle formed by each point of the grid with the original reference axis

```
angle_degrees=[[]*2]*N
for i in range(1,N+1):
innerproduct=(data_rotated['xi'][di_sorted[2][0]])*(data_rotated['xi'][i])+(data_rotated['yi'][di_sorted[2][0]])*(data_rotated['yi'][i])
   #print('  Inner product of the vector of the point ',i,' with the vector of the starting point (62,77):',innerproduct)
   angle_cos=round(innerproduct/(di[63][1]*di[i-1][1]), 4)
   angle_sin=round((data_rotated['yi'][i])/di[i-1][1], 4)
   #print(i,'cos: ',angle_cos,'sin:',angle_sin,'\n')
   angle_rad=math.acos(angle_cos)
   if angle_cos>=0 and angle_sin<=0:
      angle_degrees.insert(i-1,[i,(180*angle_rad/math.pi)])
   elif angle_cos<=0 and angle_sin<=0:
      angle_degrees.insert(i-1,[i,(180*angle_rad/math.pi)])
   elif angle_cos<=0 and angle_sin>=0:
      angle_degrees.insert(i-1,[i,(360-180*angle_rad/math.pi)])
   elif angle_cos>=0 and angle_sin>=0:
      angle_degrees.insert(i-1,[i,(360-180*angle_rad/math.pi)])
   angle_degrees.pop()
#print(angle_degrees)
print(' Sort the angles of the points forming the points with the original random axis in ascending order ','\n')
angle_degrees_sorted=sorted(angle_degrees,key=lambda l:l[1],reverse=False)
#print(angle_degrees_sorted,'\n')
```

Cluster creation based on the average number of visits of each salesman

```
cluster_id=[x for x in range(1,saleman+1)]
#print(cluster_id)
usum=0
x=1
clusters=[[]*2]*N
for i in range(1,N+1):
   usum=usum+data['ui'][angle_degrees_sorted[i-1][0]]
```

```
    if usum<=umean:
        clusters.insert(i-1,[angle_degrees_sorted[i-1][0],cluster_id[x-1]])
        clusters.pop()
    else:
        x=x+1
        clusters.insert(i-1,[angle_degrees_sorted[i-1][0],cluster_id[x-1]])
        clusters.pop()
        #print('Το άθροισμα του αριθμού των επισκέψεων για το cluster ',x-1,' είναι: ',umean)
        usum=0
print('Ομαδοποίηση των πελατών σε cluster με βάση τον μέσο αριθμό επισκέψεων','\n')
print(clusters)
x1=[]
y1=[]
x2=[]
y2=[]
x3=[]
y3=[]
x=1
for i in range(1,N+1):
    if clusters[i-1][1]==x:
        x1.append(data['xi'][clusters[i-1][0]])
        y1.append(data['yi'][clusters[i-1][0]])
    elif clusters[i-1][1]==x+1:
        x2.append(data['xi'][clusters[i-1][0]])
        y2.append(data['yi'][clusters[i-1][0]])
    elif clusters[i-1][1]==x+2:
        x3.append(data['xi'][clusters[i-1][0]])
        y3.append(data['yi'][clusters[i-1][0]])
```

Calculation of the center of gravity of each cluster separately

```
xmean_1=round(sum(x1)/len(x1), 2)
ymean_1=round(sum(y1)/len(y1), 2)
```

xmean_2=round(sum(x2)/len(x2), 2)

ymean_2=round(sum(y2)/len(y2), 2)

xmean_3=round(sum(x3)/len(x3), 2)

ymean_3=round(sum(y3)/len(y3), 2)

print(' The centre of gravity of the first or blue cluster is: ','xblue=', xmean_1,'yblue=',ymean_1, '\n')

print(' The centre of gravity of the second or green cluster is: ','xgreen=', xmean_2,'ygreen=',ymean_2, '\n')

print(' The centre of gravity of the third or purple cluster is: ','xpurple=',xmean_3,'ypurple=',ymean_3, '\n')

Calculation of distances of the individual centers of gravity of each cluster from the total centre of gravity of the cloud

*distance_01=math.sqrt((xmean-xmean_1)**2+(ymean-ymean_1)**2)*

*distance_02=math.sqrt((xmean-xmean_2)**2+(ymean-ymean_2)**2)*

*distance_03=math.sqrt((xmean-xmean_3)**2+(ymean-ymean_3)**2)*

print('Distance of cloud centre of gravity from centre of gravity 1 or blue:', distance_01, '\n')

print('Distance of cloud centre of gravity from centre of gravity 2 or green:', distance_02, '\n')

print('Distance of cloud centre of gravity from centre of gravity 3 or purple:', distance_03, '\n')

Calculation of distances between the individual cluster centers of gravity

*distance_12=math.sqrt((xmean_1-xmean_2)**2+(ymean_1-ymean_2)**2)*

*distance_13=math.sqrt((xmean_1-xmean_3)**2+(ymean_1-ymean_3)**2)*

*distance_23=math.sqrt((xmean_2-xmean_3)**2+(ymean_2-ymean_3)**2)*

print('Distance between center of gravity 1 or blue and center of gravity 2 or green:', distance_12, '\n')

print('Distance between center of gravity 1 or blue and center of gravity 3 or purple:', distance_13, '\n')

print('Distance between center of gravity 2 or green and center of gravity 3 or purple:', distance_23, '\n')

We make the following lists to join the cluster centers of gravity together

xmean_all = [xmean_1, xmean_2, xmean_3,xmean_1]

ymean_all = [ymean_1, ymean_2, ymean_3,ymean_1]

Make the following lists to join the centers of gravity with the total

xmean_all_initial=[xmean_1,xmean,xmean_2,xmean,xmean_3]

ymean_all_initial=[ymean_1,ymean,ymean_2,ymean,ymean_3]

plt.scatter(x1,y1,c='blue')

plt.scatter(x2,y2,c='green')

plt.scatter(x3,y3,c='purple')

plt.scatter(xmean,ymean,c='black')

plt.scatter(xmean_1,ymean_1,c='red')

plt.scatter(xmean_2,ymean_2,c='red')

plt.scatter(xmean_3,ymean_3,c='red')

plt.scatter(initial_x,initial_y,c='yellow')

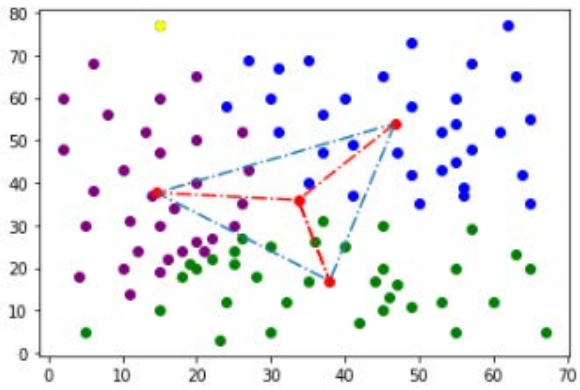
plt.title("Clustering step_2 with initial point 64 (15,77)")

plt.plot(xmean_all, ymean_all, '-.')

plt.plot(xmean_all_initial, ymean_all_initial, '-.',c='red')

plt.show()

Up to this point we get the result shown in the following cutout



Step 3: Problem of assigning customers to salesman

In this step we divide the customer points into 3 clusters based on the equal distribution of turnover by solving the mathematical model developed in the above chapter 5.

Calculate an assignment cost table to each vendor based on the clustering of step 3 with zeros with units. Place 0 when the customer is placed in the same cluster as the one in step 2 and 1 if not.

*c_ik=[[]*3]*(3*N+3)*

print(clusters)

counter=0

```
for i in range(0, N):
    if clusters[i][0]==1:
        if clusters[i][1]==1:
            c_ik[clusters[i][0]]=[clusters[i][0],1,0]
            c_ik[clusters[i][0]+1]=[clusters[i][0],2,1]
            c_ik[clusters[i][0]+2]=[clusters[i][0],3,1]
        if clusters[i][1]==2:
            c_ik[clusters[i][0]]=[clusters[i][0], 1, 1]
            c_ik[clusters[i][0]+1]=[clusters[i][0], 2, 0]
            c_ik[clusters[i][0]+2]=[clusters[i][0], 3, 1]
        if clusters[i][1]==3:
            c_ik[clusters[i][0]]=[clusters[i][0], 1, 1]
            c_ik[clusters[i][0]+1]=[clusters[i][0], 2, 1]
            c_ik[clusters[i][0]+2]=[clusters[i][0], 3, 0]
    else:
        if clusters[i][1]==1:
            c_ik[3*clusters[i][0]-2]=[clusters[i][0], 1, 0]
            c_ik[3*clusters[i][0]-1]=[clusters[i][0], 2, 1]
            c_ik[3*clusters[i][0]]=[clusters[i][0], 3, 1]

        if clusters[i][1]==2:
            c_ik[3*clusters[i][0]-2]=[clusters[i][0], 1, 1]
            c_ik[3*clusters[i][0]-1]=[clusters[i][0], 2, 0]
            c_ik[3*clusters[i][0]]=[clusters[i][0], 3, 1]
        if clusters[i][1]==3:
            c_ik[3*clusters[i][0]-2]=[clusters[i][0], 1, 1]
            c_ik[3*clusters[i][0]-1]=[clusters[i][0], 2, 1]
            c_ik[3*clusters[i][0]]=[clusters[i][0], 3, 0]

c_ik.pop(-2)
c_ik.pop(-1)
```

```
c_ik.pop(0)

print(c_ik)
```

Calculation of the table of assignment costs to each salesman based on the clustering of step 2 with the distances of the points from the centers of gravity of the individual clusters.

```
xmean_all_clusters = [xmean_1, xmean_2, xmean_3]

ymean_all_clusters = [ymean_1, ymean_2, ymean_3]

c_ik_2 = {}

for i in range(1,N+1):
   idict={}
   for k in range(1,saleman+1):
      if c_ik[3*(i-1)+k-1][2] == 1:
         idict[k] = round(math.sqrt( (data['xi'][i] - xmean_all_clusters[k-1])**2
                                 + (data['yi'][i] - ymean_all_clusters[k-1])**2 ) , 3)
      else:
         idict[k] = 0
   c_ik_2[i]=idict
print(c_ik_2)
```

We display the cost in a dictionary so that we can incorporate it into the model that we will hit next.

```
odict={}

cluster_vima_3={}

for i in range(1,N+1):
   idict={}
   for k in range(1,saleman+1):
      idict[k]=c_ik[3*(i-1)+k-1][2]


      if c_ik[3*(i-1)+k-1][2] == 0:
         cluster_vima_3[i] = k
   odict[i]=idict
print(odict)

print(cluster_vima_3)
```

Creating and solving a mathematical model using Gurobi

*from gurobipy import **

Data of the problem

v_T=0.95

v_V=0.95

#The quantity Q has been calculated above and has been called qmean. For the problem data it equals 1029 for each salesman

#The turnover q_i of each customer is a data of the problem and is mentioned above as data['qi']

#The required frequency of visits is a given of the problem and has been calculated above as a function of turnover

#The average number of visits for each customer has a computer above oh umean and equals 58

Create a dictionary for the second limitation

data_dict=data.to_dict()

Create a model

m = Model('Vima_3')

Binary decision variable x_{ik}, which takes the value 1 when customer i is assigned to seller k and 0 if "no

x_ik=m.addVars([(i,k) for i in range(1,N+1) for k in range(1,saleman+1)], vtype=GRB.BINARY, name='x_ik')

Objective function: minimize the cost of outsourcing to salesman

*m.setObjective(quicksum(c_ik_2[i][k] * x_ik[i, k] for i in range(1,N+1) for k in range(1,saleman+1)),GRB.MINIMIZE)*

Set of constraints 1: guarantees that each customer will be assigned to one salesman

m.addConstrs((quicksum(x_ik[i, k] for k in range(1,saleman+1)) == 1 for i in range (1,N+1)),name='con_1')

Set of Constraints 2: The total turnover of each seller will not be outside the prescribed limits

*m.addConstrs((quicksum(x_ik[i, k]*data_dict['qi'][i]*data_dict['ui'][i] for i in range(1,N+1)) >= v_T*qmean for k in range(1,saleman+1)),name='prwtos_klados_2')*

*m.addConstrs((quicksum(x_ik[i, k]*data_dict['qi'][i]*data_dict['ui'][i] for i in range(1,N+1))<= (2-v_T)*qmean for k in range(1,saleman+1)),name='deuteros_klados_2')*

Set of constraints 3: The total number of visits by each seller will not be outside the prescribed limits

*m.addConstrs((quicksum(x_ik[i, k]*data_dict['ui'][i] for i in range(1,N+1)) >= v_V*umean for k in range(1,saleman+1)),name='prwtos_klados_3')*

*m.addConstrs((quicksum(x_ik[i, k]*data_dict['ui'][i] for i in range(1,N+1)) <= (2-v_V)*umean for k in range(1,saleman+1)),name='deuteros_klados_3')*

Optimization

m.optimize()

print('Number of variables: ', m.numVars)

Create a new dictionary concerning the clustering of clients after the optimization of step 3

```
c_ik_After = []

c_ik_After = m.getAttr('x', x_ik.values())

cluster_vima_4={}

odict2={}

for i in range(1,N+1):

   idict={}

   for k in range(1,saleman+1):

      idict[k]=c_ik_After[3*(i-1)+k-1]


      if c_ik_After[3*(i-1)+k-1] == 1:

         cluster_vima_4[i] = k

   odict2[i]=idict

print(cluster_vima_4)
```

Print results

```
for i in range(1, N+1):

   if cluster_vima_3[i] != cluster_vima_4[i]:

      print("The client ", i, " is moved from the cluster ",cluster_vima_3[i], " of step 3 to the cluster ",cluster_vima_4[i]," of step 4 ")
```

Illustration of the above solutions into a plot

```
x1_vima4=[]

y1_vima4=[]

x2_vima4=[]

y2_vima4=[]
```

```
x3_vima4=[]
y3_vima4=[]

x=1
for key in cluster_vima_4.keys():
    if cluster_vima_4[key]==x:
        x1_vima4.append(data['xi'][key])
        y1_vima4.append(data['yi'][key])
    elif cluster_vima_4[key]==x+1:
        x2_vima4.append(data['xi'][key])
        y2_vima4.append(data['yi'][key])
    elif cluster_vima_4[key]==x+2:
        x3_vima4.append(data['xi'][key])
        y3_vima4.append(data['yi'][key])
```

Calculation of the center of gravity of each cluster separately

```
xmean_1_vima4=round(sum(x1_vima4)/len(x1_vima4), 2)
ymean_1_vima4=round(sum(y1_vima4)/len(y1_vima4), 2)

xmean_2_vima4=round(sum(x2_vima4)/len(x2_vima4), 2)
ymean_2_vima4=round(sum(y2_vima4)/len(y2_vima4), 2)

xmean_3_vima4=round(sum(x3_vima4)/len(x3_vima4), 2)
ymean_3_vima4=round(sum(y3_vima4)/len(y3_vima4), 2)
```

We make the following lists to join the cluster centers of gravity together

```
xmean_all_vima4 = [xmean_1_vima4, xmean_2_vima4, xmean_3_vima4, xmean_1_vima4]
ymean_all_vima4 = [ymean_1_vima4, ymean_2_vima4, ymean_3_vima4, ymean_1_vima4]
```

Make the following lists to join the centers of gravity with the total

```
xmean_all_initial_vima4 = [xmean_1_vima4, xmean, xmean_2_vima4, xmean, xmean_3_vima4]
```

ymean_all_initial_vima4 = [ymean_1_vima4, ymean, ymean_2_vima4, ymean, ymean_3_vima4]

plt.scatter(x1_vima4, y1_vima4, c='blue')

plt.scatter(x2_vima4, y2_vima4, c='green')

plt.scatter(x3_vima4, y3_vima4, c='purple')

plt.scatter(xmean, ymean, c='black')

plt.scatter(xmean_1_vima4 ,ymean_1_vima4, c='red')

plt.scatter(xmean_2_vima4, ymean_2_vima4 ,c='red')

plt.scatter(xmean_3_vima4, ymean_3_vima4, c='red')

plt.scatter(initial_x,initial_y,c='yellow')

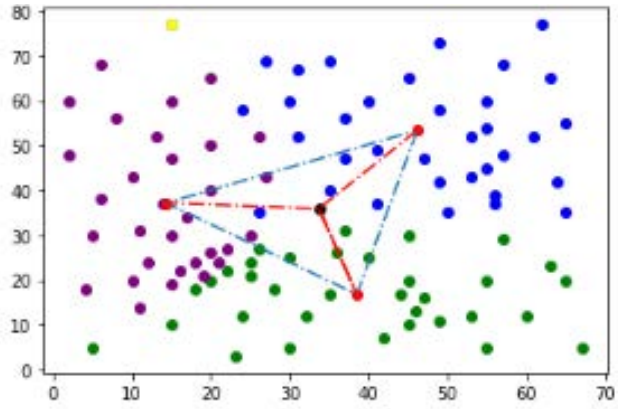
plt.title("Clustering step_4 with initial point 64 (15,77)")

plt.plot(xmean_all_vima4, ymean_all_vima4, '-.')

plt.plot(xmean_all_initial_vima4, ymean_all_initial_vima4, '-.',c='red')

plt.show()

Up to this point we get the result shown in the following cutout



Step 4: Problem transferring customers from salesman to salesman

In this step we try to correct the overlaps that may have arisen from the above step by solving the mathematical model developed in chapter 5.

Useful Data

Calculation of the total turnover generated for each salesman in the previous (step 3)

qk_total={1: 0, 2: 0, 3: 0}

uk_total={1: 0, 2: 0, 3: 0}

for i in range(1, N+1):

*qk_total[cluster_vima_4[i]] = qk_total[cluster_vima_4[i]] + data_dict['qi'][i] * data_dict['ui'][i]*

```
    uk_total[cluster_vima_4[i]] = uk_total[cluster_vima_4[i]] + data_dict['ui'][i]
print(qk_total)
print(uk_total)
q_s = {}
u_s = {}
num_of_transitions=0
```

Comparison of the dictionaries we made from step 2 and step 3 to find out for which customers there is a transfer request

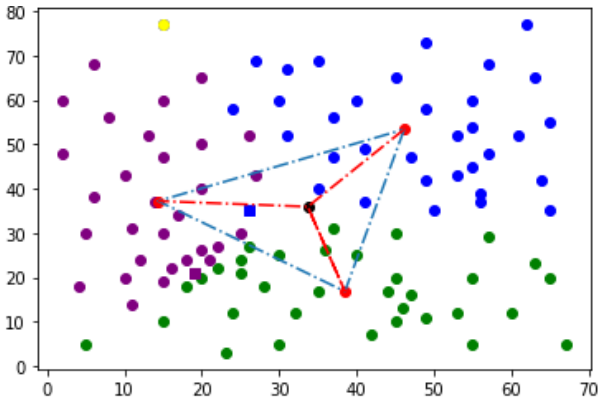
```
for i in range(1, N+1):
    if cluster_vima_3[i] != cluster_vima_4[i]:
        num_of_transitions = num_of_transitions + 1
        print("Ο πελάτης ", i, " μετατίθεται από το cluster ",cluster_vima_3[i], " του βήματος 3 στο cluster ",cluster_vima_4[i]," του βήματος 4 ")
        q_s[i] = data_dict['qi'][i]*data_dict['ui'][i]
        print("Ο πελάτης ", i, " θέλει φορτίο / ποσότητα  q_s ", q_s)
        u_s[i] = data_dict['ui'][i]
        print("Ο πελάτης ", i, " έχει συχνότητα επίσκεψησ u_s ", u_s)

print("We have  ", num_of_transitions, "total transfer requests")
```

Display of all points on a graph together with the points requesting a transfer

```
plt.scatter(x1_vima4, y1_vima4, c='blue')
plt.scatter(x2_vima4, y2_vima4, c='green')
plt.scatter(x3_vima4, y3_vima4, c='purple')
plt.scatter(xmean, ymean, c='black')
plt.scatter(xmean_1_vima4 ,ymean_1_vima4, c='red')
plt.scatter(xmean_2_vima4, ymean_2_vima4 ,c='red')
plt.scatter(xmean_3_vima4, ymean_3_vima4, c='red')
plt.scatter(initial_x,initial_y,c='yellow')
plt.title("Clustering step_5 with initial point 64 (15,77)")
plt.plot(xmean_all_vima4, ymean_all_vima4, '-.')
plt.plot(xmean_all_initial_vima4, ymean_all_initial_vima4, '-.',c='red')
```

```
for i in q_s.keys():
    if cluster_vima_4[i] == 1:
        plt.scatter(data_dict['xi'][i], data_dict['yi'][i], c="blue", marker="s")
    elif cluster_vima_4[i] == 2:
        plt.scatter(data_dict['xi'][i], data_dict['yi'][i], c="green", marker="s")
    else:
        plt.scatter(data_dict['xi'][i], data_dict['yi'][i], c="purple", marker="s")


plt.show()
```

Up to this point we get the result shown in the following cutout with the points that have ready transpositions being represented by a square



Find keys which hold the maximum/minimum values in qk_total, uk_total

```
qk_total_key_max = max(qk_total, key= lambda x: qk_total[x])
qk_total_key_min = min(qk_total, key= lambda x: qk_total[x])


uk_total_key_max = max(uk_total, key= lambda x: uk_total[x])
uk_total_key_min = min(uk_total, key= lambda x: uk_total[x])


Qmax = qk_total[qk_total_key_max]
Qmin = qk_total[qk_total_key_min]


Umax = uk_total[uk_total_key_max]
Umin = uk_total[uk_total_key_min]
```

```
print("Qmax: ", Qmax, " Qmin:", Qmin)
print("Umax: ", Umax, " Umin: ", Umin)
```

Definition of the binary variable z_sD_minus={}

```
D_plus={}
for i in range(1, N+1):
    idict_minus={}
    idict_plus={}
    if cluster_vima_3[i] != cluster_vima_4[i]:
        for k in range(1, saleman+1):
            if k == cluster_vima_4[i]:
                idict_minus[k] = 1
                idict_plus[k] = 0
            elif k == cluster_vima_3[i]:
                idict_plus[k] = 1
                idict_minus[k] = 0
            else:
                idict_plus[k] = 0
                idict_minus[k] = 0

        D_minus[i]=idict_minus
        D_plus[i]=idict_plus

print("D_minus: ", D_minus)
print("D_plus: ", D_plus)
```

Create a model

```
n = Model('Vima_4')
```

Binary decision variable z_s, equal to 1 if the transfer request is accepted and equal to 0 if it is not accepted

```
z_s=n.addVars([(i) for i in q_s.keys()], vtype=GRB.BINARY, name='z_s')
```

Objective function: maximize the satisfaction of transfer requests

n.setObjective(quicksum(z_s[i] for i in q_s.keys()),GRB.MAXIMIZE)

Set of constraints 1 - Lower Limit

*n.addConstrs((quicksum((z_s[i]*D_plus[i][k]*q_s[i]*u_s[i] - z_s[i]*D_minus[i][k]*q_s[i]*u_s[i]) for i in q_s.keys()) >= Qmin - qk_total[k]*

for k in range(1,saleman+1)),name='Constraint_1_LB')

Set of constraints 1 - Upper Limit

*n.addConstrs((quicksum((z_s[i]*D_plus[i][k]*q_s[i]*u_s[i] - z_s[i]*D_minus[i][k]*q_s[i]*u_s[i]) for i in q_s.keys()) <= Qmax - qk_total[k]*

for k in range(1,saleman+1)),name='Constraint_1_UB')

Set of constraints 2 - Lower Limit

*n.addConstrs((quicksum((z_s[i]*D_plus[i][k]*u_s[i] - z_s[i]*D_minus[i][k]*u_s[i]) for i in q_s.keys()) >= Umin - uk_total[k]*

for k in range(1,saleman+1)),name='Constraint_2_LB')

Set of constraints 2 - Upper Limit

*n.addConstrs((quicksum((z_s[i]*D_plus[i][k]*u_s[i] - z_s[i]*D_minus[i][k]*u_s[i]) for i in q_s.keys()) <= Umax - uk_total[k]*

for k in range(1,saleman+1)),name='Constraint_2_UB')

Optimization

n.optimize()

Print the variables z_s[i]

for v in n.getVars():

#if v.x == 1:

print('%s %g' % (v.varName, v.x))

If the solution of step 4 does not accept the transfer requests, i.e., if the variables z_s become equal to 0, then the user has two options:

1. either choose to satisfy the absolute geographic distribution, in which case the requests of unsatisfied customers will be satisfied at the expense of the equal distribution of turnover and total number of visits, thus choosing to continue with the clustering of step 2

2. or not to choose to satisfy the absolute geographic distribution, in which case we leave the graph as it is by satisfying the constraints on turnover and number of visits, thus choosing to continue with the clustering of step 3

To continue we chose to satisfy the geographical distribution so we continue with the clystering of step 2

Step 5: Daily assignment of customers to salesmen

The first step is to divide each area of sellers into 5 clusters using the k-means algorithm

We import libraries

```
import sklearn
from sklearn.cluster import KMeans
from collections import Counter
import seaborn as sns
```

 # Determine the number of sub-clusters of each cluster in step 2

```
k = 5
```

Apply kmeans for each initial cluster in step 2

```
C1=[[]*2]*len(x1)
for i in range (1, len(x1)+1):
    C1.insert(i-1,[x1[i-1], y1[i-1]])
    C1.pop()

C2=[[]*2]*len(x2)
for i in range (1, len(x2)+1):
    C2.insert(i-1,[x2[i-1], y2[i-1]])
    C2.pop()

C3=[[]*2]*len(x3)
for i in range (1, len(x3)+1):
    C3.insert(i-1,[x3[i-1], y3[i-1]])
    C3.pop()

X1 = np.array(C1)
X2 = np.array(C2)
```

```
X3 = np.array(C3)


print(C1)
print(X1[:,0])
print(X1[:,1])

```

Apply K-means

```
kmeans_1 = KMeans(n_clusters=k, random_state=0).fit(X1)
kmeans_2 = KMeans(n_clusters=k, random_state=0).fit(X2)
kmeans_3 = KMeans(n_clusters=k, random_state=0).fit(X3)


#print(kmeans_1.labels_)
print("<--------------------------------------------------->")
#count1 = Counter(kmeans_1.labels_)
#print(count1)
#count2 = Counter(kmeans_2.labels_)
#print(count2)
#count3 = Counter(kmeans_3.labels_)
#print(count3)
print("<--------------------Finding the Number of Visits in each cluster assigned to each client ------------------>")
```

Convert pandas dataframe to list

```
data_x = data['xi'].tolist()
data_y = data['yi'].tolist()
```

Finding the indexes of saleman's customer indexes 1

```
index1_ui = []
for j in range (0, len(X1[:,0])):
   for i in range (0, N):
      if data_x[i] == X1[j,0] and data_y[i] == X1[j,1]:
         index1_ui.append(i+1)
print(index1_ui)
ui_1 = []
```

```
for i in range (0, len(index1_ui)):
    ui_1.append(data['ui'][index1_ui[i]])
```

Finding the indexes of saleman's customer indexes 2

```
index2_ui = []
for j in range (0, len(X2[:,0])):
    for i in range (0, N):
        if data_x[i] == X2[j,0] and data_y[i] == X2[j,1]:
            index2_ui.append(i+1)
print(index2_ui)
ui_2 = []
for i in range (0, len(index2_ui)):
    ui_2.append(data['ui'][index2_ui[i]])
```

Finding the indexes of saleman's customer indexes 3

```
index3_ui = []
for j in range (0, len(X3[:,0])):
    for i in range (0, N):
        if data_x[i] == X3[j,0] and data_y[i] == X3[j,1]:
            index3_ui.append(i+1)
print(index3_ui)
ui_3 = []
for i in range (0, len(index3_ui)):
    ui_3.append(data['ui'][index3_ui[i]])
print("<---------------------------------------------------->")
```

Create a numpy array containing the 'ui' + 'labels' of saleman 1

```
ui_1np   = np.array(ui_1)
labels_1 = np.array(kmeans_1.labels_)
ui_labels_1 = np.column_stack((ui_1np, labels_1))
```

Create a numpy array containing the 'ui' + 'labels' of saleman 2

```
ui_2np   = np.array(ui_2)
labels_2 = np.array(kmeans_2.labels_)
ui_labels_2 = np.column_stack((ui_2np, labels_2))
```

Create a numpy array containing the 'ui' + 'labels' of saleman 3

```
ui_3np   = np.array(ui_3)
labels_3 = np.array(kmeans_3.labels_)
ui_labels_3 = np.column_stack((ui_3np, labels_3))
print("<---------------------------------------------------->")
num_of_elem_c0=0
num_of_elem_c1=0
num_of_elem_c2=0
num_of_elem_c3=0
num_of_elem_c4=0
sum_ui_c0=0
sum_ui_c1=0
sum_ui_c2=0
sum_ui_c3=0
sum_ui_c4=0
```

Create the sub-clusters from the original cluster 1 of step 2

```
for i in range (1, len(x1)+1):
   if kmeans_1.labels_[i-1] == 0:
      num_of_elem_c0 = num_of_elem_c0 + 1
      sum_ui_c0 = sum_ui_c0 + ui_labels_1[i-1][0]
      plt.scatter(x1[i-1], y1[i-1], c='#34e543', label='Cluster_0')
   elif kmeans_1.labels_[i-1] == 1:
      num_of_elem_c1 = num_of_elem_c1 + 1
      sum_ui_c1 = sum_ui_c1 + ui_labels_1[i-1][0]
      plt.scatter(x1[i-1], y1[i-1], c='#e93fb0', label='Cluster_1')
   elif kmeans_1.labels_[i-1] == 2:
```

```
        num_of_elem_c2 = num_of_elem_c2 + 1
        sum_ui_c2 = sum_ui_c2 + ui_labels_1[i-1][0]
        plt.scatter(x1[i-1], y1[i-1], c='#c8e246', label='Cluster_2')
    elif kmeans_1.labels_[i-1] == 3:
        num_of_elem_c3 = num_of_elem_c3 + 1
        sum_ui_c3 = sum_ui_c3 + ui_labels_1[i-1][0]
        plt.scatter(x1[i-1], y1[i-1], c='#1185ee', label='Cluster_3')
    elif kmeans_1.labels_[i-1] == 4:
        num_of_elem_c4 = num_of_elem_c4 + 1
        sum_ui_c4 = sum_ui_c4 + ui_labels_1[i-1][0]
        plt.scatter(x1[i-1], y1[i-1], c='#f45633', label='Cluster_4')
  plt.title("Kmeans on Cluster 1")
plt.show()
print(" Cluster 0 - Green: ", num_of_elem_c0, " sum of visits: ", sum_ui_c0, "\n",
      "Cluster 1 – Purple : ", num_of_elem_c1, "sum of visits:", sum_ui_c1, "\n",
      "Cluster 2 - Yellow: ", num_of_elem_c2, "sum of visits: ", sum_ui_c2, "\n",
      " Cluster 3 - Blue: ", num_of_elem_c3, " sum of visits:", sum_ui_c3, "\n",
      " Cluster 4 - Orange: ", num_of_elem_c4, " sum of visits: ", sum_ui_c4, "\n"
     )


num_of_elem_c0=0
num_of_elem_c1=0
num_of_elem_c2=0
num_of_elem_c3=0
num_of_elem_c4=0
sum_ui_c0=0
sum_ui_c1=0
sum_ui_c2=0
sum_ui_c3=0
sum_ui_c4=0
```

Create the sub-clusters from the original cluster 2 of step 2

```
for i in range (1, len(x2)+1):
    if kmeans_2.labels_[i-1] == 0:
        num_of_elem_c0 = num_of_elem_c0 + 1
        sum_ui_c0 = sum_ui_c0 + ui_labels_2[i-1][0]
        plt.scatter(x2[i-1], y2[i-1], c='#34e543', label='Cluster_0')
    elif kmeans_2.labels_[i-1] == 1:
        num_of_elem_c1 = num_of_elem_c1 + 1
        sum_ui_c1 = sum_ui_c1 + ui_labels_2[i-1][0]
        plt.scatter(x2[i-1], y2[i-1], c='#e93fb0', label='Cluster_1')
    elif kmeans_2.labels_[i-1] == 2:
        num_of_elem_c2 = num_of_elem_c2 + 1
        sum_ui_c2 = sum_ui_c2 + ui_labels_2[i-1][0]
        plt.scatter(x2[i-1], y2[i-1], c='#c8e246', label='Cluster_2')
    elif kmeans_2.labels_[i-1] == 3:
        num_of_elem_c3 = num_of_elem_c3 + 1
        sum_ui_c3 = sum_ui_c3 + ui_labels_2[i-1][0]
        plt.scatter(x2[i-1], y2[i-1], c='#1185ee', label='Cluster_3')
    elif kmeans_2.labels_[i-1] == 4:
        num_of_elem_c4 = num_of_elem_c4 + 1
        sum_ui_c4 = sum_ui_c4 + ui_labels_2[i-1][0]
        plt.scatter(x2[i-1], y2[i-1], c='#f45633', label='Cluster_4')
      plt.title("Kmeans on Cluster 2")
plt.show()

print("  Cluster 0 – Green : ", num_of_elem_c0, " sum of visits: ", sum_ui_c0, "\n",
      " Cluster 1 - Purple: ", num_of_elem_c1, " sum of visits: ", sum_ui_c1, "\n",
      "Cluster 2 - Yellow: ", num_of_elem_c2, " sum of visits: ", sum_ui_c2, "\n",
      "Cluster 3 - Blue: ", num_of_elem_c3, " sum of visits: ", sum_ui_c3, "\n",
      "Cluster 4 - Orange: ", num_of_elem_c4, " sum of visits: ", sum_ui_c4, "\n"
     )
num_of_elem_c0=0
```

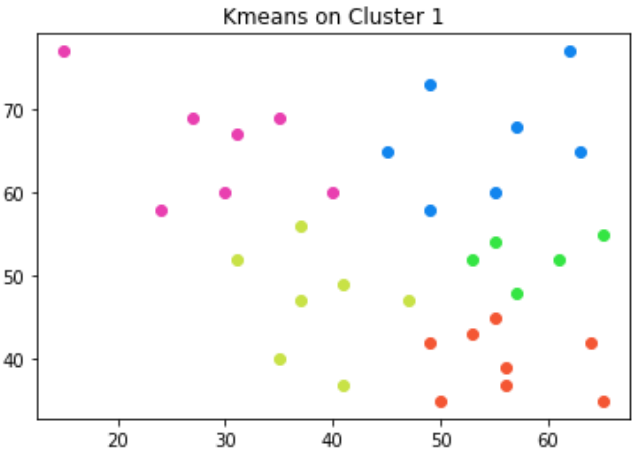
```
num_of_elem_c1=0
num_of_elem_c2=0
num_of_elem_c3=0
num_of_elem_c4=0
sum_ui_c0=0
sum_ui_c1=0
sum_ui_c2=0
sum_ui_c3=0
sum_ui_c4=0
```

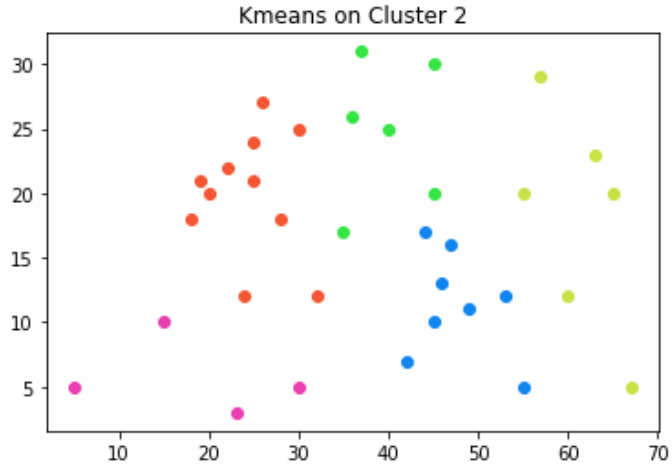
Create the sub-clusters from the original cluster 3 of step 2

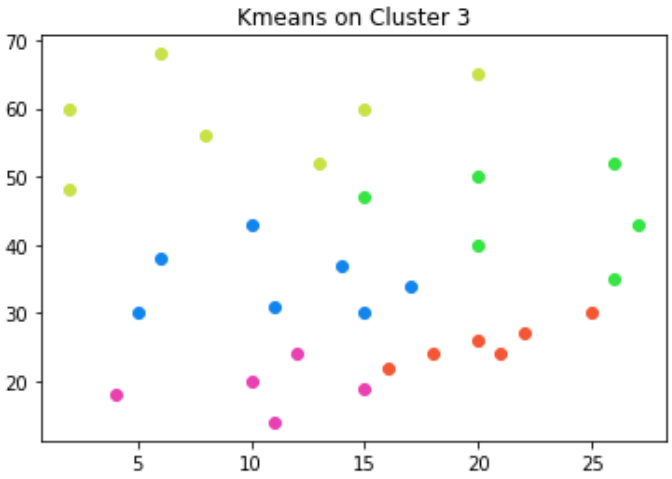
```
for i in range (1, len(x3)+1):
    if kmeans_3.labels_[i-1] == 0:
        num_of_elem_c0 = num_of_elem_c0 + 1
        sum_ui_c0 = sum_ui_c0 + ui_labels_3[i-1][0]
        plt.scatter(x3[i-1], y3[i-1], c='#34e543', label='Cluster_0')
    elif kmeans_3.labels_[i-1] == 1:
        num_of_elem_c1 = num_of_elem_c1 + 1
        sum_ui_c1 = sum_ui_c1 + ui_labels_3[i-1][0]
        plt.scatter(x3[i-1], y3[i-1], c='#e93fb0', label='Cluster_1')
    elif kmeans_3.labels_[i-1] == 2:
        num_of_elem_c2 = num_of_elem_c2 + 1
        sum_ui_c2 = sum_ui_c2 + ui_labels_3[i-1][0]
        plt.scatter(x3[i-1], y3[i-1], c='#c8e246', label='Cluster_2')
    elif kmeans_3.labels_[i-1] == 3:
        num_of_elem_c3 = num_of_elem_c3 + 1
        sum_ui_c3 = sum_ui_c3 + ui_labels_3[i-1][0]
        plt.scatter(x3[i-1], y3[i-1], c='#1185ee', label='Cluster_3')
    elif kmeans_3.labels_[i-1] == 4:
        num_of_elem_c4 = num_of_elem_c4 + 1
        sum_ui_c4 = sum_ui_c4 + ui_labels_3[i-1][0]
        plt.scatter(x3[i-1], y3[i-1], c='#f45633', label='Cluster_4')
```

```
plt.title("Kmeans on Cluster 3")
plt.show()


print(" Cluster 0 - Green: ", num_of_elem_c0, " sum of visits: ", sum_ui_c0, "\n",
      " Cluster 1 - Purple: ", num_of_elem_c1, " sum of visits: ", sum_ui_c1, "\n",
      "Cluster 2 - Yellow: ", num_of_elem_c2, " sum of visits: ", sum_ui_c2, "\n",
      "Cluster 3 - Blue: ", num_of_elem_c3, " sum of visits: ", sum_ui_c3, "\n",
      "Cluster 4 - Orange: ", num_of_elem_c4, " sum of visits: ", sum_ui_c4, "\n"
     )
print("<------------------------------------------------------------------------>")
print("\n")
```

Up to this point the 5 clusters for each vendor's area of responsibility have emerged







To avoid involving periodicity in our model so as not to increase its complexity, we choose to create virtual points for those that require multiple visits so that all my points require one visit at the end. That is, instead of having one point requiring two visits, I have two points (with the same sequences) requiring one visit, and correspondingly, instead of having one point requiring three visits, I have three points requiring one visit each.

We will then reapply the k-means algorithm with the assumption that we will not allow points with the same coordinates to be in the same cluster.

Create non-existent points for each blue, green, purple cluster. The non-existent points indicate the points need to be

created as to make the ui equal to 1 for all points.

```
blue_one_ui = [[]*2]

index_from_data = []

k=0

for i in range(1, len(blue) + 1):
    if data['ui'][blue[i-1]] == 2:
        blue_one_ui.insert(k, [data['xi'][blue[i-1]] , data['yi'][blue[i-1]]])
        index_from_data.insert(k, blue[i-1])
        blue_one_ui.insert(k+1,   [data['xi'][blue[i-1]] , data['yi'][blue[i-1]]])
        index_from_data.insert(k+1, blue[i-1])
        k = k + 2
    elif data['ui'][blue[i-1]] == 3:
        blue_one_ui.insert(k, [data['xi'][blue[i-1]] , data['yi'][blue[i-1]]])
        index_from_data.insert(k, blue[i-1])
        blue_one_ui.insert(k+1,   [data['xi'][blue[i-1]] , data['yi'][blue[i-1]]])
```

```
        index_from_data.insert(k+1, blue[i-1])
        blue_one_ui.insert(k+2,   [data['xi'][blue[i-1]] , data['yi'][blue[i-1]]])
        index_from_data.insert(k+2, blue[i-1])
        k = k + 3
    else:
        blue_one_ui.insert(k, [data['xi'][blue[i-1]] , data['yi'][blue[i-1]]])
        index_from_data.insert(k, blue[i-1])
        k = k + 1
blue_one_ui.pop()
#index_from_data.pop()

print(blue_one_ui)
print("Len of blue_one_ui:", len(blue_one_ui))
```

Kmeans for the new clusters

```
X1_new = np.array(blue_one_ui)
#print(C1)
#  εφαρμογή του Kmeans
kmeans_1_new = KMeans(n_clusters=5, random_state=0).fit(X1_new)
print("<---------- Clustering before ---------->")
print(kmeans_1_new.labels_)
num_of_elem_c0_new = 0
num_of_elem_c1_new = 0
num_of_elem_c2_new = 0
num_of_elem_c3_new = 0
num_of_elem_c4_new = 0
```

Creating sub-clusters of blue_one_ui

```
for i in range (1, len(blue_one_ui)+1):
    if kmeans_1_new.labels_[i-1] == 0:
        num_of_elem_c0_new = num_of_elem_c0_new + 1
        plt.scatter(blue_one_ui[i-1][0], blue_one_ui[i-1][1], c='#34e543', label='Cluster_0')
```

```
    elif kmeans_1_new.labels_[i-1] == 1:
        num_of_elem_c1_new = num_of_elem_c1_new + 1
        plt.scatter(blue_one_ui[i-1][0], blue_one_ui[i-1][1], c='#e93fb0', label='Cluster_1')
    elif kmeans_1_new.labels_[i-1] == 2:
        num_of_elem_c2_new = num_of_elem_c2_new + 1
        plt.scatter(blue_one_ui[i-1][0], blue_one_ui[i-1][1], c='#c8e246', label='Cluster_2')
    elif kmeans_1_new.labels_[i-1] == 3:
        num_of_elem_c3_new = num_of_elem_c3_new + 1
        plt.scatter(blue_one_ui[i-1][0], blue_one_ui[i-1][1], c='#1185ee', label='Cluster_3')
    elif kmeans_1_new.labels_[i-1] == 4:
        num_of_elem_c4_new = num_of_elem_c4_new + 1
        plt.scatter(blue_one_ui[i-1][0], blue_one_ui[i-1][1], c='#f45633', label='Cluster_4')

plt.title("Kmeans_new on Cluster 1 before")
plt.show()

print(" Cluster 0 - Green: ", num_of_elem_c0_new, "\n",
      "Cluster 1 - Purple: ", num_of_elem_c1_new, "\n",
      "Cluster 2 - Yellow: ", num_of_elem_c2_new, "\n",
      "Cluster 3 - Blue: ", num_of_elem_c3_new, "\n",
      "Cluster 4 - Orange: ", num_of_elem_c4_new, "\n")
centroids = kmeans_1_new.cluster_centers_
print("Centroid of Green cluster: ", centroids[0], "\n",
      "Centroid of Purple cluster: ", centroids[1], "\n",
      "Centroid of Yellow cluster: ", centroids[2], "\n",
      "Centroid of Blue cluster: ", centroids[3], "\n",
      "Centroid of Orange cluster: ", centroids[4], "\n",
     )
```

Calculate the distances of double and triple points with the same coordinates from the centres of gravity of the other sub-cluters to be moved to the nearest one

```
index_i_ui=[[]*2]
```

```
index_i=[]
num_of_occur=0

for i in range(1, len(blue_one_ui)):
    if(blue_one_ui[i-1][0] == blue_one_ui[i][0] and blue_one_ui[i-1][1] == blue_one_ui[i][1]):
        index_i.append(i-1)

for i in range(0, len(blue_one_ui)):
    index_i_ui.insert(i, [index_from_data[i], data['ui'][index_from_data[i]]])

index_i_ui.pop()

print("blue_one_ui")
print(blue_one_ui)
print('\n')
print("index_i_ui")
print(index_i_ui)
print('\n')
print("index_from_data")
print(index_from_data)
print('\n')
```

Find distances between points with ui = 2,3 and centroids of the sub-clusters. Also, change the label of that points

```
idx=0
m=0
dist_centroid=[]
dmin=0
labels_dmin=[]
label_transfer=0

for i in range(0, len(blue_one_ui)):
```

```
if index_i_ui[i][1] == 2 and i >= m:
    m = i + 2
    for l in range(0, 5):
        if l != kmeans_1_new.labels_[i]:
            dist_centroid.append( math.sqrt((centroids[l][0] - data['xi'][index_i_ui[i][0]])**2
                        + (centroids[l][1] - data['yi'][index_i_ui[i][0]])**2) )
            labels_dmin.append(l)
        else:
            continue

    dmin = min(dist_centroid)
    idx = dist_centroid.index(dmin)
    label_transfer = labels_dmin[idx]
    kmeans_1_new.labels_[i+1] = label_transfer
    dist_centroid.clear()
    labels_dmin.clear()

elif index_i_ui[i][1] == 3 and i >= m:
    m = i + 3
    for l in range(0, 5):
        if l != kmeans_1_new.labels_[i]:
            dist_centroid.append( math.sqrt((centroids[l][0] - data['xi'][index_i_ui[i][0]])**2
                        + (centroids[l][1] - data['yi'][index_i_ui[i][0]])**2) )
            labels_dmin.append(l)
        else:
            continue
    Find first minimum distance
    dmin = min(dist_centroid)
    idx = dist_centroid.index(dmin)
    label_transfer = labels_dmin[idx]
    kmeans_1_new.labels_[i+1] = label_transfer
```

```
        dist_centroid.remove(dmin)
        labels_dmin.remove(label_transfer)
```

Find second minimum distance

```
        dmin = min(dist_centroid)
        idx = dist_centroid.index(dmin)
        label_transfer = labels_dmin[idx]
        kmeans_1_new.labels_[i+2] = label_transfer
        dist_centroid.clear()
        labels_dmin.clear()

print("<------------------- Clustering after ------------------->")
print(kmeans_1_new.labels_)
print("\n")

num_of_elem_c0_new_after = 0
num_of_elem_c1_new_after = 0
num_of_elem_c2_new_after = 0
num_of_elem_c3_new_after = 0
num_of_elem_c4_new_after = 0
```

Create sub-clusters of blue_one_ui after transposing the double and triple points

```
for i in range (1, len(blue_one_ui)+1):
    if kmeans_1_new.labels_[i-1] == 0:
        num_of_elem_c0_new_after = num_of_elem_c0_new_after + 1
        #plt.scatter(blue_one_ui[i-1][0], blue_one_ui[i-1][1], c='#34e543', label='Cluster_0')
    elif kmeans_1_new.labels_[i-1] == 1:
        num_of_elem_c1_new_after = num_of_elem_c1_new_after + 1
        #plt.scatter(blue_one_ui[i-1][0], blue_one_ui[i-1][1], c='#e93fb0', label='Cluster_1')
    elif kmeans_1_new.labels_[i-1] == 2:
        num_of_elem_c2_new_after = num_of_elem_c2_new_after + 1
        #plt.scatter(blue_one_ui[i-1][0], blue_one_ui[i-1][1], c='#c8e246', label='Cluster_2')
```

```
    elif kmeans_1_new.labels_[i-1] == 3:
        num_of_elem_c3_new_after = num_of_elem_c3_new_after + 1
        #plt.scatter(blue_one_ui[i-1][0], blue_one_ui[i-1][1], c='#1185ee', label='Cluster_3')
    elif kmeans_1_new.labels_[i-1] == 4:
        num_of_elem_c4_new_after = num_of_elem_c4_new_after + 1
        #plt.scatter(blue_one_ui[i-1][0], blue_one_ui[i-1][1], c='#f45633', label='Cluster_4')
print(" Cluster 0 - Greeen: ", num_of_elem_c0_new_after, "\n",
      "Cluster 1 - Purple: ", num_of_elem_c1_new_after, "\n",
      "Cluster 2 - Yellow: ", num_of_elem_c2_new_after, "\n",
      "Cluster 3 - Blue: ", num_of_elem_c3_new_after, "\n",
      "Cluster 4 - Orange: ", num_of_elem_c4_new_after, "\n")
```