**Large Scale Optimization**

**-**

**Part Time**

**Team Members:**

Konstantina Georgiopoulou (p2822004)

Anastasios Theodorou (p2822007)

Christos Kallaras (p2822009)

Stavros Kasiaris (p2822022)

Date: *January 2022*

**Contents**

[**Introduction** 2](#_Toc93167544)

# **Introduction**

**1o Ερώτημα: Γράψτε σε python τις κλάσεις, πίνακες και γενικότερα την απαιτούμενη υποδομή για αναπαρασταθεί το πρόβλημα**

Το πρόβλημα αναπαριστάται ως ένα Vehicle Routing Problem(VRP).

Δημιουργήσαμε τις κλάσεις Model, Node και Route για να αναπαραστήσουμε το πρόβλημα. Στην κλάση Model υπάρχει η function Build Model όπου περιλαμβάνεται το total duration, που είναι 150 και οι συντεταγμένες, το service time, που παίρνει τιμές από 5 έως 10 καθώς και το profit, που και αυτό παίρνει τιμές από 5 έως 20, του κάθε κόμβου. Επίσης, υπολογίζεται και ο πίνακας κόστους, που στην περίπτωσή μας το κόστος ορίζεται ως η Ευκλείδεια απόσταση μεταξύ των κόμβων, δηλαδή το time matrix.

H κλάση Node, περιλαμβάνει τις συντεταγμένες, το id, το service time και το profit, η ένδειξη isRoute, που δείχνει εάν ένας κόμβος έχει συμπεριληφθεί στην διαδρομή ή όχι καθώς και το profit per time, που είναι η διαίρεση του profit δια το service time, έτσι ώστε να γνωρίζουμε το κέρδος που έχουμε για κάθε κόμβο ανά λεπτό εξυπηρέτησης και να μπορέσουμε να ταξινομήσουμε στην συνέχεια τους πιο επικερδής κόμβους να τους συμπεριλάβουμε πρώτους στις διαδρομές.

Τέλος, η κλάση Route περιλαμβάνει την αλληλουχία των κόμβων, το duration της διαδρομής που είναι 150, το profit που «παράγει» η κάθε διαδρομή και το time της κάθε διαδρομής, που δείχνει πόσο χρονικά γεμάτη είναι η κάθε διαδρομή και αλλάζει με την προσθήκη ενός νέου κόμβου.

**2ο Ερώτημα: Φτιάξτε έναν κατασκευαστικό αλγόριθμο ο οποίος θα σχηματίζει μία ολοκληρωμένη λύση**

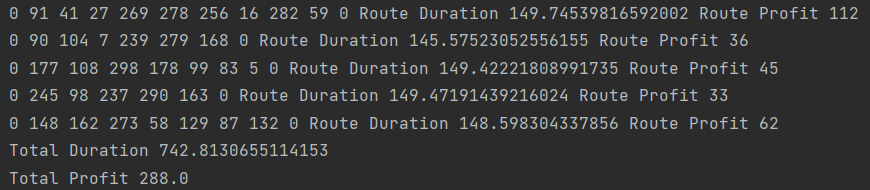
Χρησιμοποιήσαμε τον κατασκευαστικό αλγόριθμο Minimum Insertions για να δημιουργήσουμε μια λύση που θα περιλαμβάνει την αλληλουχία κόμβων 5 διαδρομών-φορτηγών και ως αντικειμενική συνάρτηση ορίζεται το συνολικό κέρδος των κόμβων που εξυπηρετούν και τα πέντε φορτηγά.

Αρχικά, θέτουμε όλους τους κόμβους ως μη- επιστεπτόμενους, θέτοντας το isRouted False, μέσα από την function: *SetRoutedFlagToFalseForAllCustomers().*

Έπειτα, μέσα από την function: *MinimumInsertions()*, δημιουργείται μια κενή λύση και ταξινομεί όλους τους πελάτες με βάση το μεγαλύτερο profit/ service time, ούτως ώστε να εξυπηρετήσει πρώτους τους πιο επικερδής κόμβους. Όσο υπάρχουν κόμβοι που δεν έχουν εξυπηρετηθεί, παίρνει τον κάθε customer και τον τοποθετεί σε κάθε σημείο των διαδρομών, εφόσον ο χρόνος της διαδρομής συν το service time του συγκεκριμένου customer το επιτρέπει(είναι κάτω του ορίου του 150). Για το καλύτερο cost added μείων cost removed, δηλαδή για τον μικρότερο χρόνο μεταβίβασης ανάμεσα στους κόμβους, προσθέτει τον συγκεκριμένο customer στην διαδρομή και υπολογίζει τον νέο χρόνο της διαδρομής (μέσω του cost added- cost removed) και το νέο profit της διαδρομής.

Στην συνέχεια, μέσω της function: *TestSolution()* για κάθε διαδρομή υπολογίζει τους χρόνους μεταβίβασης και service time των κόμβων καθώς και το profit της κάθε διαδρομής και εμφανίζει αν υπάρχει πρόβλημα με τις διαδρομές.

Τέλος, μέσω της function: *ReportSolution(),* η οποία παίρνει σαν input την λύση που δημιουργήσαμε και κάνει print την αλληλουχία των κόμβων της κάθε διαδρομής, τον χρόνο και το profit της κάθε διαδρομής αλλά και συνολικά.



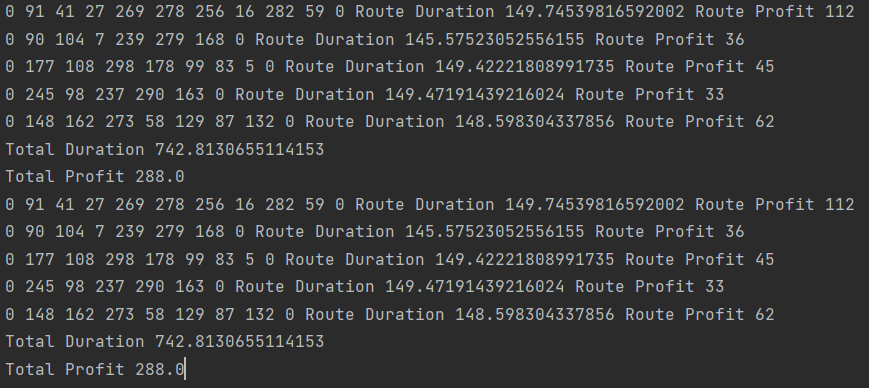
**3o Ερώτημα: Γράψτε σε Python τέσσερεις τελεστές τοπικής έρευνας**

H function: Local Search() λαμβάνει ως input την αρχική λύση που δημιουργείται από το 2ο ερώτημα και έναν αριθμό από το 0 έως το 3, ανάλογα με ποιον τελεστή θέλουμε να χρησιμοποιήσει και εμφανίζει την νέα λύση ύστερα από την εφαρμογή του τελεστή.

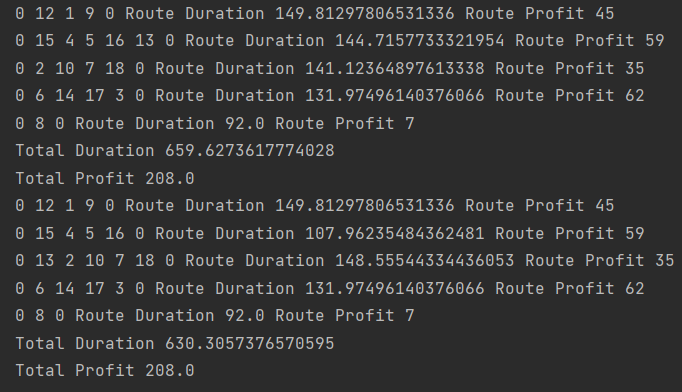
**3.1: Relocate: Επανατοποθέτηση κάθε καλυπτόμενου πελάτη σε οποιοδήποτε διαφορετικό σημείο της λύσης.**

Μέσω της Function: FindBestRelocationMove(), για κάθε κόμβο που εξυπηρετούμε και για κάθε route τον επανατοποθετούμε σε διαφορετικό σημείο της λύσης. Αν αφορά διαφορετικές διαδρομές, υπολογίζουμε τον χρόνο της targeted route συν το service time και αν είναι μεγαλύτερος του 150 πάμε στον επόμενο κόμβο. Αν δεν είναι, υπολογίζουμε το cost added, που είναι ο χρόνος για να πάμε στον νέο κόμβο συν το service time στην νέα διαδρομή και το cost removed που είναι οι χρόνοι μεταβίβασης που αφαιρούνται και το service time αυτού του κόμβου σε αυτήν την διαδρομή. **Αν και στις δύο routes τηρείται ο περιορισμός της χωρητικότητας του 150 κάνουμε Store την συγκεκριμένη κίνηση και Apply.** Είναι σημαντικό να αναφέρουμε ότι σε αυτήν την κίνηση δεν αλλάζει το profit(καθώς εξυπηρετούμε κάθε φορά τον ίδιο αριθμό κόμβων) αλλά θέλουμε να πετύχουμε όσο το δυνατόν πιο γεμάτες διαδρομές, για να μπορέσουμε στην συνέχεια να εξυπηρετήσουμε και άλλους κόμβους.

Για 300 κόμβους, βλέπουμε ότι δεν υπάρχει διαφορά(είναι ήδη πολύ γεμάτες οι διαδρομές και δεν μας επιτρέπουν αλλαγές)



Για 20 κόμβους βλέπουμε την αλλαγή στους χρόνους, αφήνοντας την δεύτερη διαδρομή πιο κενή για να δεχτεί και άλλους κόμβους:

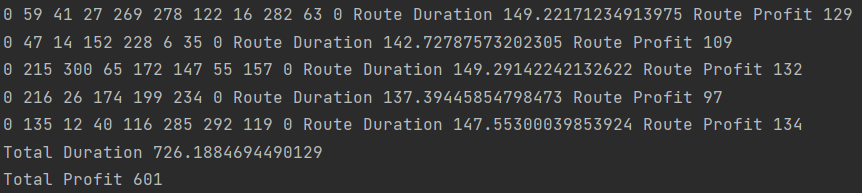


**3.2: Swap: Αντιμετάθεση των θέσεων εξυπηρέτησης οποιουδήποτε ζεύγους καλυπτόμενων πελατών.**

**3.3: Insertion: Εισαγωγή ενός οποιουδήποτε μη καλυπτόμενου πελάτη σε οποιοδήποτε σημείο της λύσης**

**3.4: Profitable Swap: Αντικατάσταση οποιουδήποτε καλυπτόμενου πελάτη με οποιονδήποτε μη καλυπτόμενο πελάτη.**

H function: *FindBestUncoveredSwapMove(),* παίρνει όλους τους μη-καλυπτόμενους πελάτες, και για κάθε διαδρομή, παίρνει τον προηγούμενο και τον επόμενο κόμβο από το σημείο της λύσης που θέλουμε να προσθέσουμε τον νέο πελάτη και υπολογίζει το cost added που είναι το κόστος μεταβίβασης συν το service time και το cost removed του κόμβου που θέλουμε να αντικαταστήσουμε. Υπολογίζεται το νέο profit και αν είναι καλύτερο από αυτό έχει τώρα η διαδρομή και τηρείται ο περιορισμός του χρόνου, **γίνεται Store η συγκεκριμένη κίνηση και Apply**



**4o Ερώτημα: . Γράψτε σε Python μία VND μέθοδο η οποία θα χρησιμοποιεί τους τελεστές του ερωτήματος (Γ) και θα βελτιώνει τη λύση του ερωτήματος (Β). Ποια είναι η τελική λύση. Ποιο το κόστος της και μετά από πόσες επαναλήψεις του αλγορίθμου δημιουργήθηκε;**