Τίτλος προγράμματος: ΜΠΣ στην Επιχειρηματική Αναλυτική (πλήρους φοίτησης)

Μάθημα: Large Scale Optimization

Διδάσκων: Ζαχαριάδης Ε.

Εργασία: Large Scale Optimization Group Assignment

Μέλη Ομάδας: Δήμος Ηλίας (**f2822102**), Τριανταφυλλάκης Ιωάννης (**f2822115**), Τσέκας Νικόλαος (**f2822116**)

Εισαγωγή

Το πρόβλημα το οποίο λύνει ο παραδοτέος κώδικας είναι η επιλογή των πελατών που θα εξυπηρετηθούν με κριτήριο την μεγιστοποίηση του κέρδους, έχοντας υπόψιν τον χρονικό περιορισμό των 150 χρονικών μονάδων που ισχύει ανά φορτηγό. Στόχος του κώδικα είναι να επιλέξει μόνο τους πλέον κερδοφόρους πελάτες (όχι απαραίτητα όλους) των οποίων όμως η εξυπηρέτηση δεν θα παραβιάσει το χρονικό περιθώριο του εκάστοτε φορτηγού. Το παρών πρόβλημα ανήκει στην κατηγορία permutation πρέπει δηλαδή να βρεθεί μια σειρά πελατών ανά διαδρομή, καθώς και στην κατηγορία selection καθώς πρέπει να επιλεχθούν πελάτες με βάσει το κέρδος που θα αποδώσουν. Ο κώδικας στην αρχική λύση επιστρέφει 5 κλειστές διαδρομές, έχοντας χρησιμοποιήσει την λογική minimum insertions για ένα Vehicle Routing Problem 5 οχημάτων, καθώς επίσης και την λογική επίλυσης ενός knapsack προβλήματος. Στην συνέχεια, δημιουργούνται 2 λύσεις (5 κλειστών διαδρομών η κάθε μία) χρησιμοποιώντας τους τελεστές Relocation και Swap οι οποίες στοχεύουν στη μείωση του συνολικού κόστους. Η τελευταία λύση που παρουσιάζεται είναι οι πέντε κλειστές διαδρομές που προέρχονται από τη χρήση μιας VND μεθόδου που χρησιμοποιεί και τους δύο τελεστές τοπικής έρευνας με στόχο την περαιτέρω μείωση του συνολικού κόστους.

Μεθοδολογία

Ο κώδικας αποτελείται από 4 αρχεία Python. Στο αρχείο *"VRP_Model"*, υπάρχουν τα classes *Node(), Route()* και *Model()*. Τα 2 πρώτα χρησιμοποιούνται για την αρχικοποίηση node objects (δηλαδή πελατών) και route objects (διαδρομών των φορτηγών) αντίστοιχα μέσα στο class *Model()* το οποίο αναπαριστά και το πρόβλημα. Συνολικά δημιουργούνται 300 πελάτες, μια αποθήκη (όπου ξεκινούν και καταλήγουν τα φορτηγά), καθώς και ένας πίνακας 301x301 με τις ευκλείδειες αποστάσεις μεταξύ όλων των πιθανών συνδυασμών πελατών και αποθήκης. Στο αρχείο *"Solver"* βρίσκεται ο κώδικας που επιλύει το πρόβλημα. Το method *IdentifyMinimumCostInsertion()*, βρίσκει σε κάθε Iteration τον πελάτη που εάν μπει στην διαδρομή, το total cost αυξάνεται κατά το λιγότερο δυνατό (Minimum Insertions), αλλά και έχει τον μεγαλύτερο λόγο κέρδους/κόστους ώστε να μεγιστοποιήσει όσο το δυνατόν περισσότερο το συνολικό κέρδος της λύσης (Knapsack).

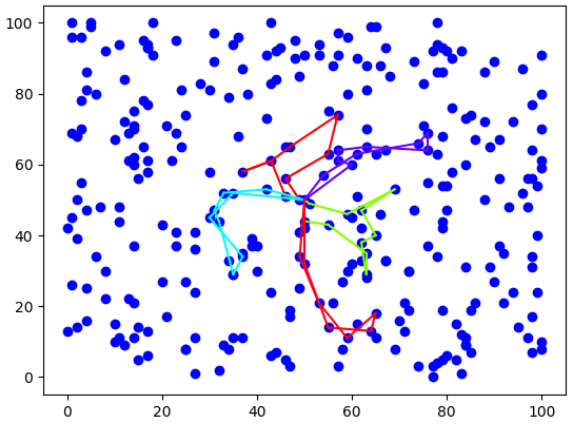
Μετά την εξαγωγή της πρώτης λύσης εφαρμόζεται κάθε φορά ένας τελεστής τοπικής έρευνας ώστε να βελτιστοποιηθεί το συνολικό κόστος. Στην αρχική λύση αρχικά εφαρμόζεται ο τελεστής relocation και στην συνέχεια πάλι στην αρχική λύση εφαρμόζεται ο τελεστής swap. Επιπλέον, γράφτηκε ένα VND method, το οποίο χρησιμοποιεί 2 operators με απώτερο στόχο την περαιτέρω μείωση του κόστους όλων των διαδρομών χρησιμοποιώντας αυτή την φορά και τους δύο τελεστές. Ο πρώτος τελεστής λέγεται *RelocationMove()*, και γράφτηκε class για αυτόν. Αυτός ο τελεστής επανατοποθετεί κάθε καλυπτόμενο πελάτη σε οποιοδήποτε διαφορετικό σημείο της λύσης, δηλαδή μπορεί να του αλλάξει σειρά στο ίδιο route αλλά μπορεί και να του αλλάξει και το ίδιο το route στο οποίο ανήκει. Ο δεύτερος τελεστής λέγεται *SwapMove()* και είναι και αυτός class. Ο συγκεκριμένος τελεστής αντιμεταθέτει τις θέσεις εξυπηρέτησης οποιουδήποτε ζεύγους καλυπτόμενων πελατών. Το αρχείο *SolutionDrawer* περιλαμβάνει methods που παράγουν plots για την λύση, ενώ το αρχείο Main καλεί το method *BuildModel()* του class *Model()* μέσα στο method *solve()* του class *Solver()* και δίνει την λύση στο terminal.

Παρουσίαση Αποτελεσμάτων

Κατά την εκτέλεση του κώδικα, ο πρώτος πίνακας του output περιλαμβάνει τις 5 κλειστές διαδρομές, το κόστος κάθε μιας, το κέρδος κάθε μιας, καθώς επίσης και το συνολικό κόστος και κέρδος. Ο πρώτος λοιπόν πίνακας περιέχει την αρχική λύση και είναι αποτέλεσμα της χρήσης των μεθόδων minimum insertions (VRP) και knapsack. Τα αποτελέσματα των διαδρομών καθώς και η γραφική τους απεικόνιση παρουσιάζονται στον Πίνακα 1 και στο Διάγραμμα 1.

```
0 67 78 86 71 188 131 43 170 130 0 Cost: 144.38 Profit: 148
0 292 260 113 185 52 154 215 152 11 30 0 Cost: 144.07 Profit: 146
0 163 206 84 226 116 248 265 231 171 0 Cost: 141.6 Profit: 118
0 133 13 111 60 108 136 217 128 22 0 Cost: 148.65 Profit: 114
0 46 238 291 205 213 266 19 0 Cost: 149.35 Profit: 104
Total Cost: 728.0615648177675 Total Profit: 630.0
```

Πίνακας 1: Αρχική Λύση προβλήματος

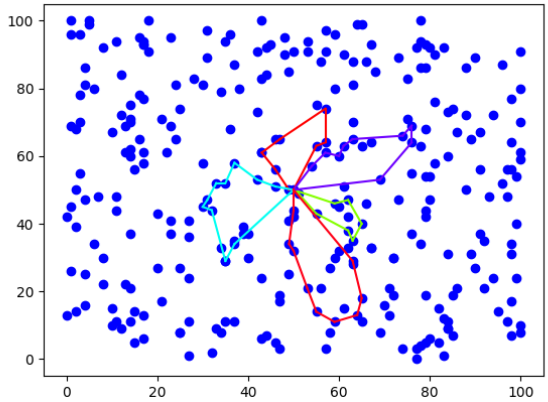


Διάγραμμα 1: Οι 5 διαδρομές της αρχικής λύσης

Στην συνέχεια, εφαρμόζονται οι προαναφερθέντες τελεστές στην αρχική λύση για την μείωση του συνολικού κόστους. Αρχική εφαρμόζεται ο τελεστής relocation. Τα αποτελέσματα των διαδρομών καθώς και η γραφική τους απεικόνιση μετά την εφαρμογή του relocation παρουσιάζονται στον Πίνακα 2 και στο Διάγραμμα 2. Όπως παρατηρείται η λύση έχει βελτιωθεί σημαντικά καθώς το κόστος μειώθηκε από 728.06 σε 649.83. Επίσης από το Διάγραμμα 2 φαίνεται ότι η μορφή των διαδρομών είναι καλύτερη. Η συγκεκριμένη λύση παράχθηκε μετά από 21 iteration.

```
Relocation Move
0 130 170 78 71 188 131 111 43 0 Cost: 118.81 Profit: 124
0 30 260 52 113 154 215 11 0 Cost: 91.4 Profit: 94
0 84 226 116 248 206 265 231 163 86 171 67 0 Cost: 147.15 Profit: 150
0 22 13 133 128 60 217 136 108 152 0 Cost: 143.14 Profit: 125
0 185 205 213 291 266 238 19 46 292 0 Cost: 149.33 Profit: 137
Total Cost: 649.83245112071 Total Profit: 630.0
```

Πίνακας 2: Λύση μετά την εφαρμογή του relocation move.

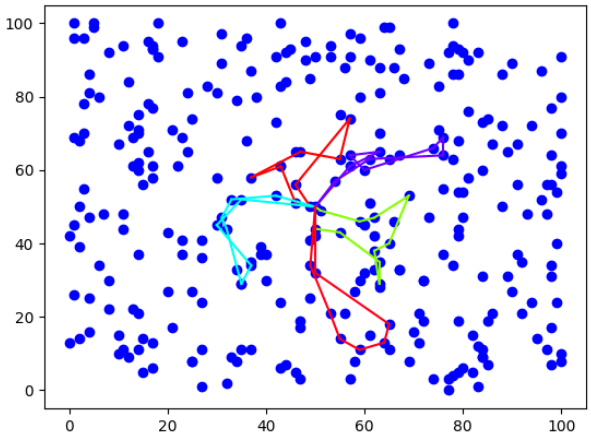


Διάγραμμα 2: Οι 5 διαδρομές της λύσης μετά το relocation move.

Τα αποτελέσματα των διαδρομών καθώς και η γραφική τους απεικόνιση μετά την εφαρμογή του swap παρουσιάζονται στον Πίνακα 3 και στο Διάγραμμα 3. Και εδώ, η λύση έχει βελτιωθεί σημαντικά καθώς το κόστος μειώθηκε από 728.06 (Αρχική λύση) σε 649.83. Η συγκεκριμένη λύση παράχθηκε μετά από 17 iteration.

```
Swap Move
0 67 71 78 86 188 43 131 170 130 0 Cost: 134.71 Profit: 148
0 292 260 113 185 52 154 152 215 11 30 0 Cost: 141.02 Profit: 146
0 163 206 84 116 226 248 265 231 171 0 Cost: 133.6 Profit: 118
0 13 60 111 133 217 136 108 128 22 0 Cost: 107.48 Profit: 114
0 46 238 266 291 213 205 19 0 Cost: 120.92 Profit: 104
Total Cost: 637.7253897024258 Total Profit: 630.0
```

Πίνακας 3: Λύση μετά την εφαρμογή του swap move

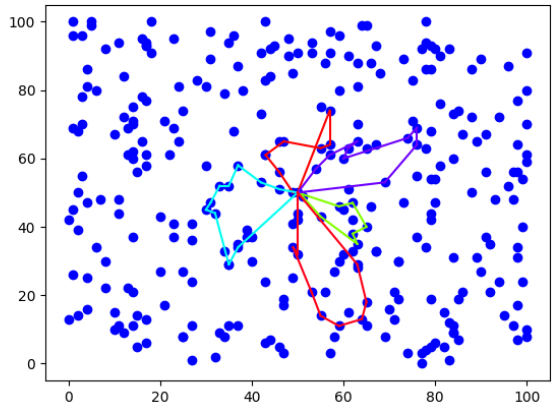


Διάγραμμα 3: Οι 5 διαδρομές της λύσης μετά το swap move

Τέλος, με την χρήση του VND χρησιμοποιούνται και οι δύο τελεστές μαζί. Να σημειωθεί ότι δεν χρησιμοποιούνται εναλλάξ, αλλά πρώτα χρησιμοποιείται ο RelocationMove() και όταν αυτός πάψει να προσφέρει μείωση στο κόστος τότε μόνο χρησιμοποιείται ο SwapMove(), καθώς μετά από δοκιμές διαπιστώσαμε πως έτσι επιτυγχάνεται ακόμα χαμηλότερο total cost με το παρόν seed(). Στον Πίνακα 4 και στο Διάγραμμα 4 παρουσιάζονται τα αποτελέσματα μετά την εφαρμογή της μεθόδου VND.

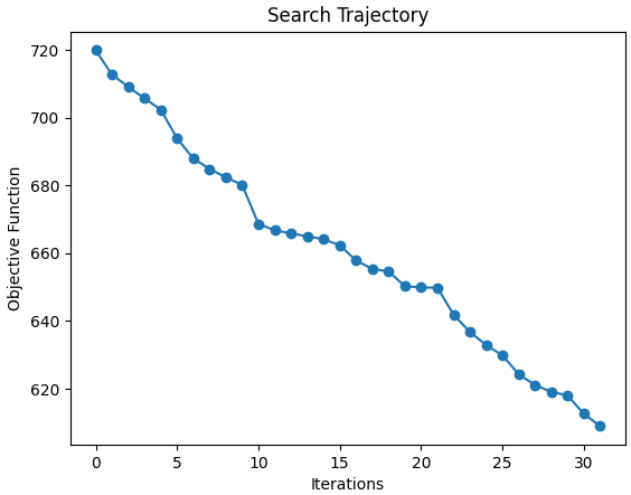
```
0 130 170 71 78 188 43 111 131 0 Cost: 106.68 Profit: 124
0 30 260 113 52 154 215 11 0 Cost: 86.33 Profit: 94
0 84 116 226 248 206 265 231 163 86 171 67 0 Cost: 139.15 Profit: 150
0 22 13 60 128 133 217 136 108 152 0 Cost: 131.37 Profit: 125
0 185 205 213 291 266 238 46 19 292 0 Cost: 145.48 Profit: 137
Total Cost: 609.0132704698824 Total Profit: 630.0
```

Πίνακας 4: Λύση μετά την εφαρμογή της μεθόδου VND



Διάγραμμα 4: Οι 5 διαδρομές της λύσης μετά τη μέθοδο VND

Όπως παρατηρείται με τη μέθοδο VND επιτυγχάνεται η μέγιστη μείωση κόστους. Συγκεκριμένα, από το κόστος 728.06 της αρχικής λύσης η παρούσα έχει μειωθεί κατά περίπου 119 μονάδες με τελικό κόστος λύσης 609.01. Όπως φαίνεται και στο Διάγραμμα 5 η παραπάνω λύση παράχθηκε μετά από 31 iterations καθώς στο σημείο αυτό η λύση παγιδεύεται σε τοπικό ελάχιστο.



Διάγραμμα 5: Τιμή αντικειμενικής συνάρτησης ανά επανάληψη