Σχεδιασμός και Βελτιστοποίηση της Εφοδιαστικής Αλυσίδας
Αναφορά Λύσης Multi-Trip VRP

Ονοματεπώνυμο:

Πρόβλημα

Το πρόβλημα που καλούμαστε να λύσουμε είναι ένα πρόβλημα δρομολόγησης οχημάτων (Vehicle Routing Problem ή VRP) με πολλαπλές επιστροφές στην αποθήκη (MultiTrip) και χωρικότητα οχήματος (Capacitated). Θεωρούμε ότι το όχημα ξεκινάει γεμάτο από την αποθήκη και έχει χωρητικότητα 160 μονάδες. Μία γενικότερη μορφή του προβλήματος αποτελεί το Travelling Salesman Problem.

- Στόχος

  Στόχος αποτελεί το να περάσει το όχημα από κάθε πελάτη (κόμβο), οι οποίοι είναι 50 και μια αποθήκη Depot, στις λιγότερες χρονικές μονάδες, ξεκινώντας και καταλήγοντας στον κόμβο 1 αποθήκη.

- Περιορισμοί

  1. Να μην παραμείνει το όχημα πάνω από 200 μονάδες χρόνου στην διαδρομή (εκτός της αποθήκης)
  2. Το όχημα να περιέχει αρκετά προιόντα για να εξυπηρετήσει τους κόμβους που επισκέπτεται εξ' ολοκλήρου

Κάθε κόμβος έχει συγκεκριμένη ζήτηση, η οποία δίνεται από τον πίνακα demand και ο χρόνος που χρειάζεται το όχημα για να μετακινηθεί από τον κόμβο Α στον κόμβο Β εκφράζεται σε χρονικές μονάδες ως η ευκλείδεια απόσταση των καρτεσιανών συντεταγμένων των εν λόγω κόμβων. Θεωρούμε ότι έχουμε στην διάθεσή μας ένα όχημα, ενώ ταυτόχρονα ορίζονται τα εξής χρονικά penalties:

1. Χρόνος εξυπηρέτησης κόμβου ίσος με 10, ο οποίος εκφράζει τον χρόνο που χρειάζεται το προσωπικό του οχήματος για να εξυπηρετήσει οποιονδήποτε κόμβο
2. Χρόνος φόρτωσης ίσος με 10, ο οποίος εκφράζει τον χρόνο που χρειάζεται το όχημα για να ανεφοδιαστεί αφού φτάσει στην αποθήκη.

Αλγόριθμος

Για την επίλυση του προβλήματος έγινε χρήση του μεθευρετικού αλγορίθμου GRASP (Greedy Randomized Adaptive Search Algorithm). Ο αλγόριθμος αυτός βασίζεται στην βελτιστοποίηση μέσω τοπικών αναζητήσεων μιας άπληστης (greedy) , με κάποιο βαθμό τυχαιότητας, αρχικής λύσης. Ο αλγόριθμος έχει δύο φάσεις:

- Construction Phase

  Για την κατασκευή της αρχικής λύσης γίνεται χρήση του αλγορίθμου πλησιέστερου γείτονα (Nearest Neighbour). Ο αλγόριθμος του πλησιέστερου γείτονα επιλέγει ως επόμενο κόμβο, εκείνον που απέχει τις λιγότερες χρονικές μονάδες από τον κόμβο που βρισκόμαστε. Ο GRASP αλγόριθμος όμως, εφόσον βασίζεται σε κάποια τυχαιότητα πρέπει να κατασκευάσει μία λίστα πιθανών επόμενων κόμβων (Restricted Candidate List) και να επιλέξει τυχαία από αυτούς τον επόμενο κόμβο.

Συνεπώς, από κάθε κόμβο εξάγουμε το Restricted Candidate List μεγέθους 4, το οποίο περιέχει τους 4 κοντινότερους κόμβους, εφικτούς προς επίσκεψη, και επιλέγει τυχαία έναν από αυτούς ως τον επόμενο κόμβο. ‘Ενας κόμβος είναι εφικτός προς επίσκεψη εάν ικανοποιεί τους περιορισμούς του προβλήματος και δεν έχει ξαναπεράσει το όχημα από αυτόν. Εάν οι περιορισμοί δεν ικανοποιούνται τότε το όχημα επιστρέφει στην αποθήκη για ανεφοδιασμό, άρα στην διαδρομή προστίθενται ο κόμβος της αποθήκης 1.
Τέλος, αφού κατασκευαστεί η αρχική λύση, δηλαδή βρεθεί μία διαδρομή με την οποία το όχημα περνάει από κάθε κόμβο, υπολογίζεται ο χρόνος που χρειάζεται το όχημα για να την εκτελέσει. Η παραπάνω λογική υλοποιείται στην συνάρτηση *initialRoute()* του κώδικα. Ο ψευδοκώδικας που την υλοποιεί είναι ο παρακάτω:

```
TOR = 0
TTS = 0
totalTime = 0
route = [1]
While ! passedFromEveryNode
        nodeNow = route(end)
        nodesToGo = nodeNow.distanceMatrix
        For nodesAlreadyVisited
                nodesToGo.remove(nodesAreadyVisited)
        nodePicked = False
        While ! nodePicked
                RCL = CloserNodes(nodesToGo, 4)
                nodeToGo = uniRandom(RCL)
                if TOR + distanceMatrix(nodeNow, nodeToGo) + distanceMatrix(1,
                nodeToGo) + servTime < timeLimit
                &&
                capacityLeft - demand(nodeToGo) >0
                        route.add(nodeToGo)
                        update(TTS,TOR,totalTime)
                        nodePicked = True
                Else
                        nodesToGo.remove(nodeToGo)
        passedEveryNode = check(route)
```

- Local Search Phase with 2-opt

Αφού πλέον έχουμε την αρχική λύση, μπορούμε να εκτελέσουμε τοπικές αναζητήσεις ελαχίστου σε διάφορες γειτονιές του προβλήματος. Για την τοπική αναζήτηση εφαρμόζεται αρχικά ο αλγόριθμος αναζήτησης 2-opt. Σύμφωνα με αυτό τον αλγόριθμο τοπικής αναζήτησης εξετάζεται αν η εναλλαγή 2 οποιονδήποτε ακμών της διαδρομής καταλήγει σε εφικτή και επιτυχή διαδρομή με μικρότερο συνολικό χρονικό κόστος (ή απλά κόστος) σε σχέση με την αρχική.

Για την διαδικασία εναλλαγής ακμών εφαρμόστηκε ο ψευδοκώδικας του wikipedia στην συνάρτησης *swap(route, a, b).* Ο ψευδοκώδικας είναι ο εξής:

```
Function newRoute = swap(route, aNodeIdx, bNodeIdx)
    newRoute = route(start : a)
    newRoute = [newRoute flip(route(aNodeIdx+1 : bNodeIdx))]
    newRoute = [newRoute route(bNodeIdx+1 : end)]
```

Με την παραπάνω διαδικασία μπορούμε να εναλλάξουμε τις ακμές από και προς τους κόμβους a και b και να επανασυνδέσουμε κατάλληλα την διαδρομή. Ο ψευδοκώδικας της τοπικής αναζήτησης είναι ο παρακάτω και υλοποιείται στην συνάρτηση *Opt2()* του κώδικα.

```
route = initialRoute
currentBestCost = initialPathCost
Start:
For i=2:end-1
    For j=i+1:end-1
        newRoute = swap(route, i, j)
        newCost = calcCost(newRoute)
        If newCost < currentBestCost
            route = newRoute
            currentBestCost = newCost
            goto START
```

Όταν πλέον δεν μπορούμε να μειώσουμε περαιτέρω το κόστος, άρα θα εκτελεστούν πλήρως και τα δύο for loops, τότε θα έχουμε εξετάσει κάθε 2-opt κίνηση και θα έχουμε πετύχει ένα τοπικό ελάχιστο του κόστους στην *currectBestCost* για την διαδρομή *route.*

- *Local Search with 1-1 Exchange*

  Ο αλγόριθμος τοπικής αναζήτησης 1-1 Exchange βασίζεται στην εναλλαγή δύο nodes της διαδρομής και την επανασύνδεση της διαδρομής. Για κάθε πιθανό 1-1 Exchange move παρατηρούμε εάν η νέα διαδρομή είναι feasible και αν βελτιώνει το κόστος. Στην θετική περίπτωση κρατάμε την νέα διαδρομή και επανεκκινούμε τον αλγόριθμο. Ο ψευδοκώδικας της παραπάνω λογικής είναι:

```
route = initialRoute
currentBestCost = initialPathCost
Start:
For i=2:end-1
    For j=i+1:end-1
        newRoute(i) = route(j)
        newRoute(j) = route(i)
        newCost = calcCost(newRoute)
        If newCost < currentBestCost
            route = newRoute
            currentBestCost = newCost
            goto START
```

Στο τέλος της παραπάνω διαδικασίας θα έχουμε στην διάθεσή μας την τοπικά ελάχιστη χρονικά διαδρομή στην μεταβλητή route και το κόστος της στην μεταβλητή

currentBestCost. Η παραπάνω λογική βρίσκεται στην συνάρτηση *oneToOneExchange()* του κώδικα.

Για τον υπολογισμό του κόστους της εκάστοτε διαδρομής και τον έλεγχο εφικτότητας κατασκευάστηκε σχετική λογική στην συνάρτηση *routeCost()*. Η λογική αυτή είναι η εξής:

```
Cost = 0
itemsLeft = capacity
TOR = 0
For node i in route
    If route(i+1) != 1
        If
        (TOR + distanceMatrix(nodeNow, nodeToGo) +  distanceMatrix(1,
        nodeToGo) + servTime < timeLimit
        &&
        capacityLeft - demand(nodeToGo) >0
        )
            capacityLeft -= demand(route(i+1))
        Else
            Cost = inf
            break;
        Cost += distanceMatrix(route(i), route(i+1)) + servTime
        TOR += distanceMatrix(route(i), route(i+1)) + servTime
    Else
        If TOR + distanceMatrix(route(i),route(i+1)) >= timeLimit
            Cost = inf
            break;
        capacityLeft = capacity
        TOR = 0
        Cost += distanceMatrix(route(i), route(i+1)) + loadTime
```

Λόγω της τυχαιότητας κατασκευής της αρχικής λύσης ο αλγόριθμος GRASP, κάθε φορά που εκτελείται θα πετυχαίνει διαφορετικό τοπικό ελάχιστο κόστους. Συνεπώς, αποτελεί λογικό να εκτελέσουμε την παραπάνω διαδικασία του construction phase και local search phase αρκετές φορές και να κρατήσουμε την διαδρομή με τον μικρότερο κόστος. Ο ψευδοκώδικας για την παραπάνω διαδικασία είναι ο εξής:

```
For i=1:5000
    route = constructInitialSol(nodes, distanceMatrix, demand, constraints)
    optRoute = localSearchAlg(route)
    optCost = calcCost(optRoute)
    If optCost < bestCost
        bestRoute = route
        bestCost = optCost
```

Στο τέλος λοιπόν των 5000 επαναλήψεων θα έχουμε την βέλτιστη διαδρομή στην μεταβλητή bestRoute και το κόστος της στην bestCost. Η παραπάνω λογική κώδικα βρίσκεται στο *main.m* αρχείο.

Αποτελέσματα

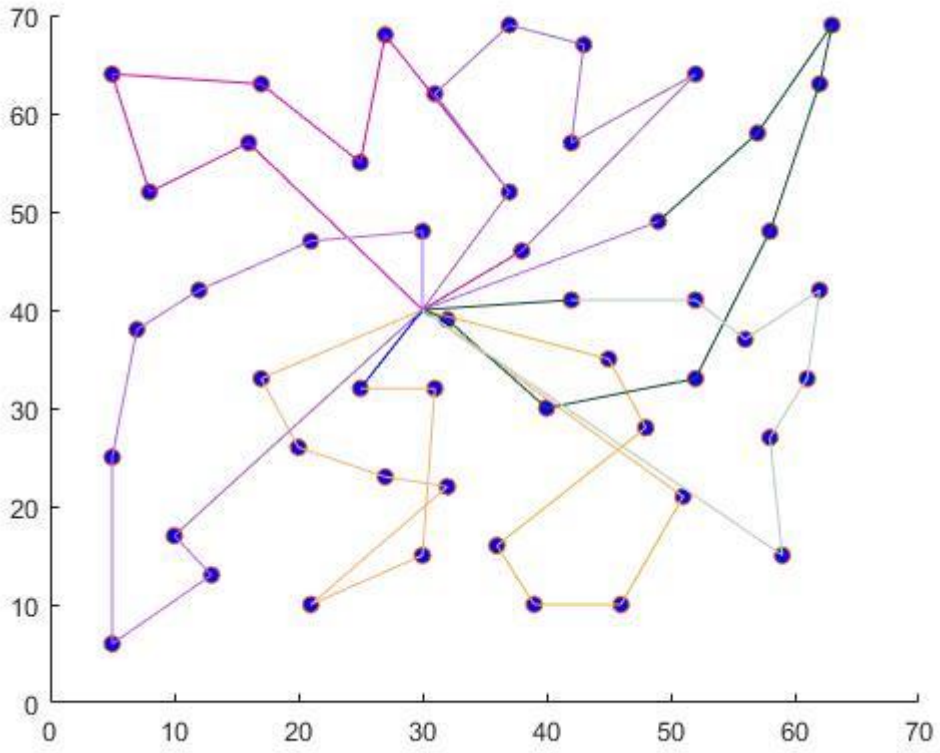
Για την επιβεβαίωση της λειτουργίας των επαναληπτικών εκτελέσεων του GRASP κατασκευάστηκε διάγραμμα που δείχνει το τοπικό ελάχιστο που πετυχαίνει ο GRASP και συνεχώς βελτιώνει. Κατά περίπτωση αλγορίθμου τοπικής αναζήτησης, τα αποτελέσματα είναι τα παρακάτω:

- GRASP with 2-opt

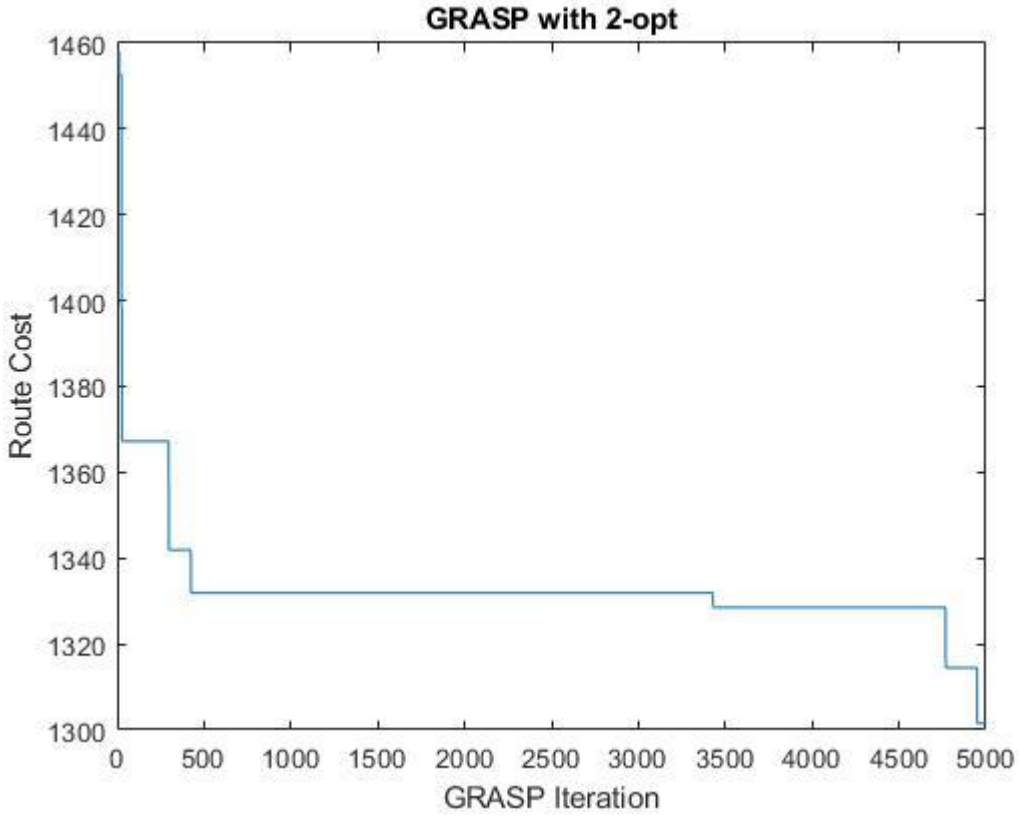
  Η βέλτιστη διαδρομή που υπολογίστηκε μετά από 5000 επαναλήψεις του construction και local search phase είναι η εξής:

  [1 48 13 45 43 38 18 5 19 1 11 34 46 16 50 39 1 2 27 49 8 44 25 24 1 33 4 23 29 32 9 2 1 42 20 41 14 26 15 7 28 1 3 21 37 36 30 10 6 47 1 12 17 51 22 35 31 40 1]

  Παρακάτω παρουσιάζεται η διαδρομή σε scatter plot. Κάθε υποδιαδρομή από την αποθήκη ως την επόμενη επίσκεψη στην αποθήκη παρουσιάζεται με διαφορετικό χρώμα.



  Η απόδοση του GRASP ως η μείωση του κόστους ανά επανάληψη φαίνεται παρακάτω:

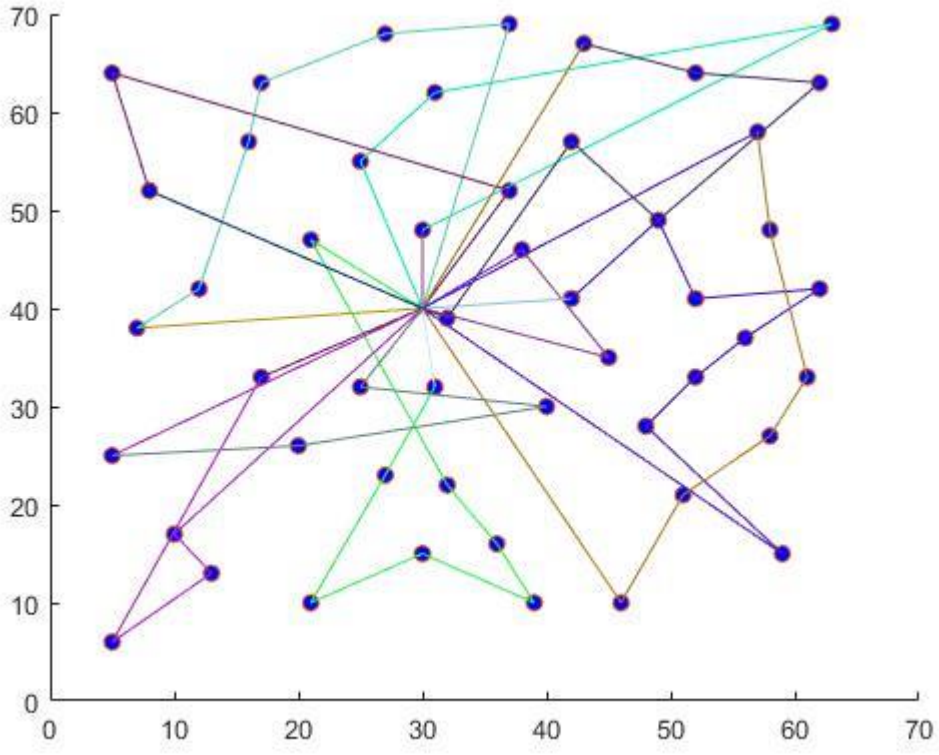


Παρατηρούμε λοιπόν ότι με την χρήση του 2-opt πετυχαίνουμε βέλτιστο κόστος 1301.51, ενώ κατά μέσο όρο οι επαναλήψεις του GRASP προτείνουν διαδρομές με κοστος 1332.85.

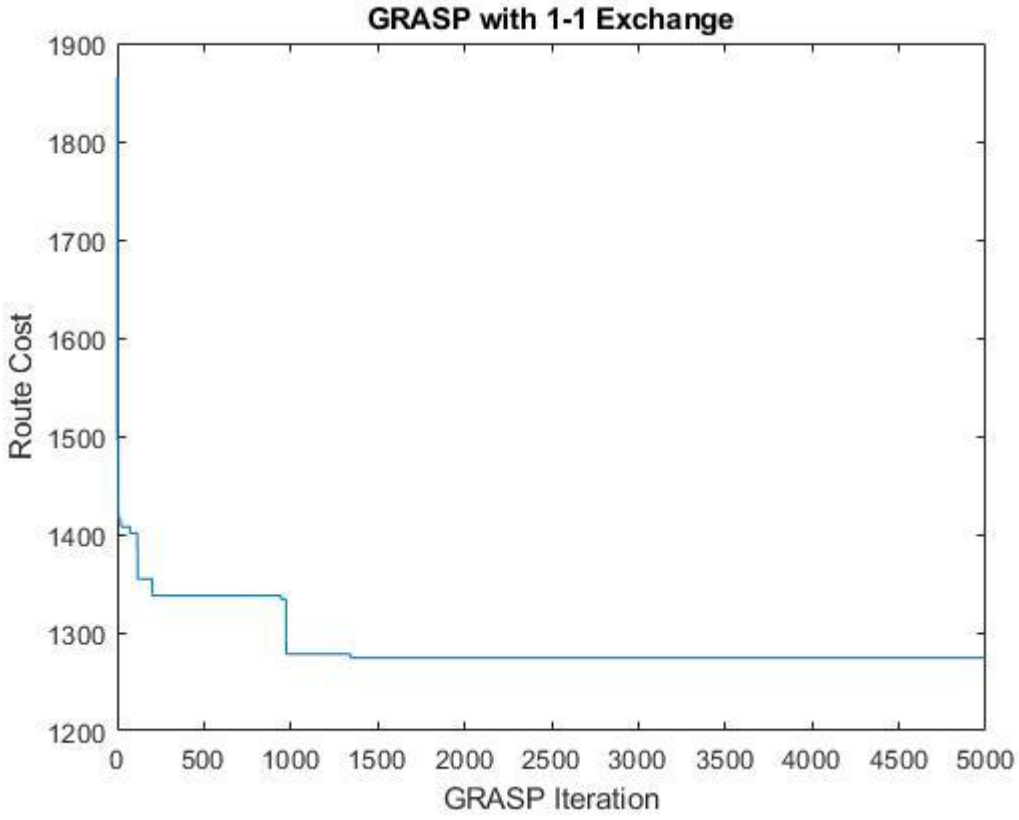
- <u>GRASP with 1-1 Exchange</u>
  Η βέλτιστη διαδρομή που υπολογίστηκε μετά από 5000 επαναλήψεις του construction και local search phase είναι η εξής:

  [1 13 48 5 18 38 45 16 46 34 47 1 33 9 32 29 27 8 24 28 1 6 39 50 31 17 22 35 10 12 1 7 19 43 20 41 42 14 1 2 23 4 37 36 21 3 1 30 51 40 11 1 49 44 25 26 15 1]

  Παρακάτω παρουσιάζεται η διαδρομή σε scatter plot.



Η απόδοση του GRASP ως η μείωση του κόστους ανά επανάληψη φαίνεται παρακάτω:



Παρατηρούμε λοιπόν ότι με την χρήση του 1-1 Exchange πετυχαίνουμε βέλτιστο κόστος 1274.21, ενώ κατά μέσο όρο οι επαναλήψεις του GRASP προτείνουν διαδρομές με κοστος 1288.86.

Γενικές Παρατηρήσεις

Γενικά παρατηρούμε ότι και οι δύο αλγόριθμοι τοπικής αναζήτησης πετυχαίνουν σχεδόν όμοιες τιμές κόστους. Αξίζει να παρατηρήσουμε ότι μετά τις πρώτες 500-1000 επαναλήψεις του GRASP το ελάχιστο κόστος αλλάζει ελάχιστα. Η πληθώρα επαναλήψεων όμως, δεν είναι γενικά εφικτή κατά την εκτέλεση του GRASP, μιας και για προβλήματα περισσότερων κόμβων οι τοπικές αναζητήσεις είναι αρκετά κοστοβόρες. Στην συγκεκριμένη περίπτωση, όμως το κόστος εκτέλεσης είναι μηδαμινό. Ούτως ή άλλος, παρατηρούμε ότι μετά από ελάχιστες αρχικές επαναλήψεις το βέλτιστο κόστος τείνει ισχυρά προς την τελική τιμή.

Μία ακόμα αξιόλογη παρατήρηση είναι ότι η τοπική αναζήτηση με 1-1 Exchange πετυχαίνει μικρότερο κόστος από την 2-Opt.