**Traveling Salesman Problem**

Τελική εργασία του μαθήματος ‘Γραμμική & Συνδυαστική βελτιστοποίηση’

Στάικος Θεόδωρος 1066578

1 Περιγραφή του προβλήματος

To πρόβλημα του πλανόδιου πωλητή εκφράζεται ως:

Έστω ένας πλήρως συνδεδεμένο δίκτυο πόλεων που απέχουν μεταξύ τους συγκεκριμένες αποστάσεις. Ποιος είναι ο βέλτιστος δρόμος, δηλαδή ακολουθία πόλεων, με την ελάχιστη συνολική απόσταση διαδρομής που πρέπει να ακολουθήσει ένας πλανόδιος πωλητής για να επισκεφθεί όλες τις πόλεις και να επιστρέψει στην αφετηρία.

Το πρόβλημα του πλανόδιου πωλητή χωρίζεται σε 3 βασικές κατηγορίες:

1. Symmetric TSP: Οι πόλεις έχουν ανά δύο ίδιο κόστος μετακίνησης από τη μία στην άλλη ανεξάρτητα της κατεύθυνσης. Δηλαδή ο πίνακας αποστάσεων είναι συμμετρικός.

Ειδίκευση του sTSP είναι η περίπτωση που οι κόμβοι βρίσκονται σε δισδιάστατο πεδίο, όπου το κόστος υπολογίζεται ως ευκλείδεια απόσταση μεταξύ των σημείων και το πρόβλημα κατηγοριοποιείται ως 2D-plane ή Euclidean TSP.

1. Asymmetric TSP: Αν υπάρχει έστω και ένα ζεύγος πόλεων που έχει διαφορετικό κόστος μετακίνησης από τη μία στην άλλη ανάλογα την κατεύθυνση τότε το πρόβλημα είναι μη-συμμετρικό TSP.
2. mTSP: Αφορά m πωλητές που έχουν κοινή ή ξεχωριστή αφετηρία και πρέπει να καλύψουν ένα δίκτυο πόλεων με βέλτιστο τρόπο, επιστρέφοντας στην αφετηρία

Στα πλαίσια της εργασίας μου, θα ασχοληθώ με το συμμετρικό πρόβλημα sTSP.

Το TSP είναι πρόβλημα συνδυαστικής βελτιστοποίησης, δηλαδή ανήκει στην οικογένεια προβλημάτων εύρεσης της βέλτιστης λύσης ενός προβλήματος μέσα από ένα σετ λύσεων πεπερασμένου πλήθους.

Έχοντας συνολικά n κόμβους – πόλεις, το πλήθος των πιθανών λύσεων είναι , άρα είναι προφανές πως η brute force επίλυση του προβλήματος δεν είναι εφικτή για υψηλό αριθμό κόμβων. Απαιτούνται, λοιπόν, αλγόριθμοι που σε πρώτη φάση απορρίπτουν από το σετ λύσεων μεγάλο αριθμό λύσεων και μειώνουν τις πιθανές διαδρομές και δευτερευόντως αλγόριθμοι χαμηλής χρονικής πολυπλοκότητας που βρίσκουν ικανοποιητική λύση χωρίς να εγγυώνται, όμως, βελτιστότητα της τελικής λύσης.

2 Πραγματικές εφαρμογές του προβλήματος

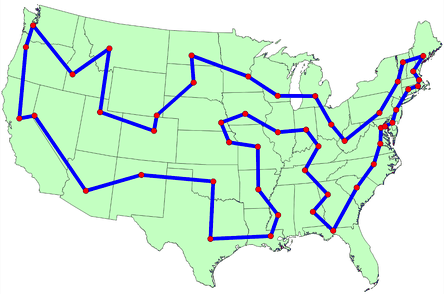
Γενικά, ως TSP μοντελοποιούνται πληθώρα προβλημάτων και εφαρμογών στη βιομηχανία.

Αλγόριθμοι επίλυσης sTSP χρησιμοποιούνται για δρομολόγηση οχημάτων ταχυδρομείου και μεταφορικών εταιριών, καθώς και για κατασκευή διαδρομών φορτηγών πλοίων και κρουαζιέρων, όπου η σύνδεση με το πρόβλημα TSP είναι προφανής.

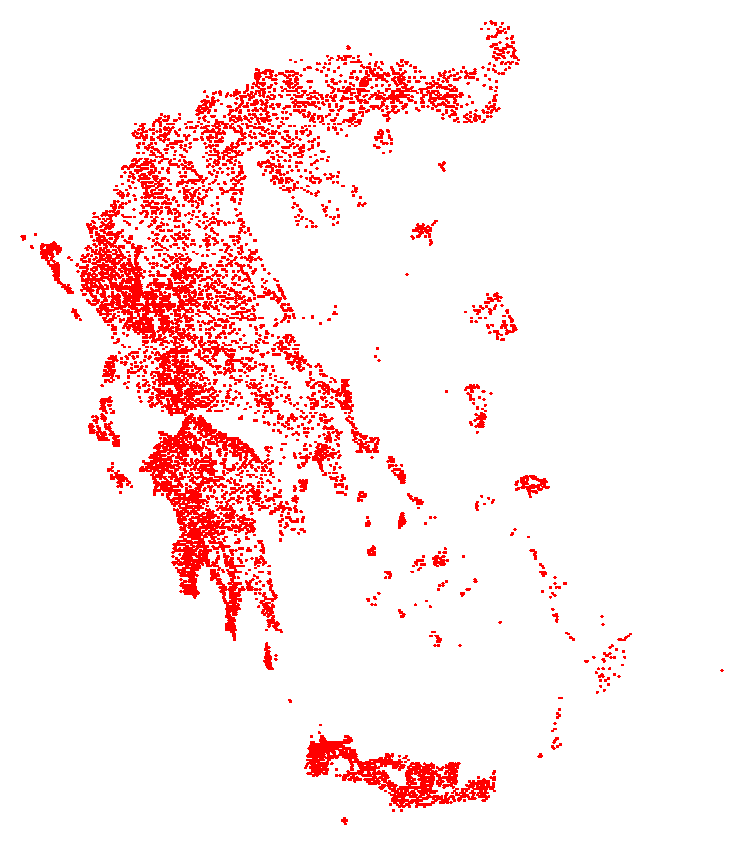
Η μοντελοποίηση προβλημάτων ως TSP problems συμβαίνει και στη βιομηχανία των ημιαγωγών για βέλτιστη κατασκευή μασκών πυριτίου, για ελάχιστη διαδρομή της κεφαλής εργαλείου κατά την διάτρηση μεταξύ επιπέδων πλακών πυριτίου και βέλτιστη καλωδίωση των παραπάνω πλακών ώστε να αποφεύγεται η περιττή χρήση υλικού.

Το εύρος εφαρμογής αλγορίθμων επίλυσης TSP είναι μεγάλο. Μερικές ακόμα εφαρμογές αφορούν την βέλτιστη χρήση και εναλλαγή αισθητήρων στην κρυσταλλογραφία έως την βέλτιστη τοποθέτηση διαφημιστικών πινακίδων για μεγιστοποίηση της επιτυχίας μιας διαφημιστικής καμπάνιας.

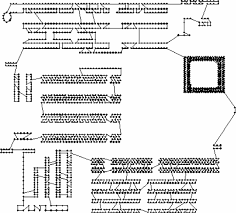
Να σημειωθεί πως τα προβλήματα TSP ταυτίζονται με τα προβλήματα προγραμματισμού μηχανής (machine scheduling programs) και η μοντελοποίηση και επίλυσή της είναι ακριβώς ίδια.



Εικόνα 1: TSP 48 Πολιτειών



Εικόνα 2: TSP 9,882 πόλεων της Ελλάδας



Εικόνα 3: TSP in PCB Drilling

3 Μέθοδοι επίλυσης του TSP

Στα πλαίσια της εργασίας μελέτησα 4 μεθόδους για επίλυση του προβλήματος του πλανόδιου πωλητή. Παρακάτω είναι μια σύντομη επεξήγηση αυτών. Περισσότερες λεπτομέρειες για τις μεθόδους επίλυσης και ανάλυση των αποτελεσμάτων παρουσιάζονται στα αντίστοιχα κεφάλαια της αναφοράς.

1. Brute Force Method:

Εύρεση όλων των πιθανών κυκλικών διαδρομών με μια συγκεκριμένη αφετηρία. Επιλέγεται η διαδρομή που έχει το μικρότερο συνολικό κόστος. Έχοντας συνολικά n κόμβους – πόλεις, το πλήθος των πιθανών λύσεων είναι , άρα είναι προφανές πως η brute force επίλυση του προβλήματος δεν είναι κατάλληλη για υψηλό αριθμό κόμβων.

1. Ακέραιος προγραμματισμός:

Με κατάλληλη μοντελοποίηση του προβλήματος TSP (περιγράφεται παρακάτω) μπορούμε να χρησιμοποιήσουμε εργαλεία επίλυσης προβλημάτων ακεραίου προγραμματισμού.

Θεωρητικά, η επίλυση με branch & bound έχει περίπου την ίδια πολυπλοκότητα με την μέθοδο Brute Force, αλλά με χρήση τεχνικών προ-επεξεργασίας και με κατάλληλη επιλογή διακλαδώσεων, οι σύγχρονοι solvers μειώνουν σημαντικά τον χρόνο επίλυσης.

1. Repetitive Nearest Neighbor:

Πρόκειται για απλή προσεγγιστική μέθοδο που υπολογίζει τη διαδρομή με βάση τον κοντινότερο κόμβο της τρέχουσας πόλης (που δεν είναι ήδη στην διαδρομή). Είναι μια greedy προσέγγιση επίλυσης του προβλήματος με χαμηλή πολυπλοκότητα Ο(n2) και σε έναν γράφο με συγκρίσιμα costs μεταξύ κόμβων μπορεί να δώσει μια ικανοποιητική λύση.

Για περαιτέρω βελτίωση της λύσης μπορούμε να εφαρμόσουμε τον αλγόριθμο με αρχή όλους της κόμβους (ή με ένα υποσύνολο που επιλέγεται με τυχαίο τρόπο) και να επιλέξουμε τη λύση με το μικρότερο συνολικό κόστος.

1. Genetic Algorithm:

Για βελτίωση της λύσης που δίνει ο αλγόριθμος Repetitive-NN εφάρμοσα γενετικό αλγόριθμο που προσπαθεί να βρει λύσεις με μικρότερο κόστος διαδρομής από τον αρχικό (δηλαδή την λύση που προκύπτει από τον αλγόριθμο του Nearest-Neighbor) κάνοντας εναλλαγές δύο πόλεων στη διαδρομή. Μοιάζει αρκετά με την προσέγγιση του [simulated annealing](https://toddwschneider.com/posts/traveling-salesman-with-simulated-annealing-r-and-shiny/), με κύρια διαφορά ότι η δική μου προσέγγιση διαχειρίζεται πληθυσμό λύσεων αντί να προσπαθεί να βελτιώσει ένα μοναδικό αντικείμενο λύσης.

4 Επίλυση TSP με ακέραιο προγραμματισμό

Έστω n πόλεις με πόλη αφετηρίας την πόλη 1.

Έχω της δυϊκές μεταβλητές απόφασης xij που υποδηλώνουν διαδρομή από την πόλη i προς την πόλη j.

Παίρνουν τιμή 1 αν η σύνδεση μεταξύ i-j ανήκει στη διαδρομή του πωλητή και τιμή 0 αν δεν ανήκει.

Έχω τον συμμετρικό C(n x n) πίνακα που δίνει το κόστος μετακίνησης μεταξύ των κόμβων – πόλεων. Τα διαγώνια στοιχεία προφανώς δεν έχουν κάποιο νόημα αφού δεν μπορεί να πραγματοποιηθεί μετακίνηση προς την τρέχουσα πόλη.

Η αντικειμενική συνάρτηση είναι το συνολικό κόστος της διαδρομής της λύσης με βάση τις μεταβλητές απόφασης. Για κάθε σύνδεση έχω κόστος όπου το κόστος της διαδρομής από πόλη i σε πόλη j.

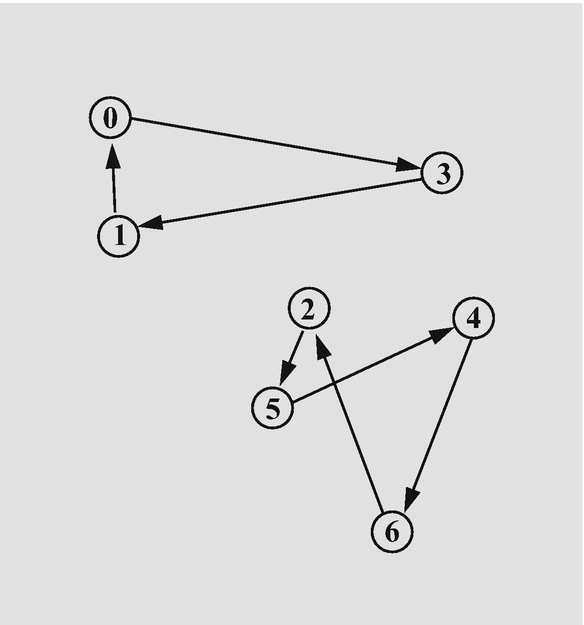
Άρα έχω πρόβλημα ελαχιστοποίησης

Οι πρώτοι δύο περιορισμοί του μοντέλου αφορούν την επίσκεψη της κάθε πόλης μόνο μία φορά.

Περιορισμός 1ος: Άφιξη σε κάθε πόλη ακριβώς μία φορά

Περιορισμός 2ος: Αναχώρηση από κάθε πόλη ακριβώς μία φορά

Ο 3ος περιορισμός είναι πιο πολύπλοκος και αφορά την μη-ύπαρξη εσωτερικών κυκλικών διαδρομών. Υπάρχουν λύσεις που συμμορφώνονται στους δύο πρώτους περιορισμούς, αλλά δεν είναι εφικτές στην πραγματικότητα αφού υπάρχουν αποκομμένοι κύκλοι στην διαδρομή.



Εικόνα 4: Αποκομμένη κυκλική διαδρομή

Ο 3ος περιορισμός εκφράζεται ως:

Με χρήση βοηθητικών μεταβλητών μπορώ να εξαλείψω τις κυκλικές διαδρομές ως εξής:

Αν τότε (Ο κόμβος i υστερεί κατά 1 θέση του κόμβου j).  
Ουσιαστικά, οι μεταβλητές εκφράζουν την σειρά επίσκεψης των αντίστοιχων κόμβων σε μία διαδρομή.  
Για παράδειγμα, αν έχω τη διαδρομή 1 -> 3 -> 2 τότε .

Έτσι, δεν μπορούν να δημιουργηθούν εσωτερικοί κύκλοι, καθώς η σειρά των βοηθητικών μεταβλητών δεν θα συμβαδίζει με τον κανόνα.  
Η παραπάνω σχέση προφανώς δεν είναι γραμμική, για αυτό εισάγουμε τον περιορισμό με την παρακάτω μορφή.

Περισσότερα για τους περιορισμούς αφαίρεσης αποκομμένων κυκλικών διαδρομών σε προβλήματα δρομολόγησης: [A note on the lifted Miller-Tucker-Zemlin subtour elimination constraints for routing problems with time windows](https://hal.archives-ouvertes.fr/hal-02947086/document)

Και ο 4ος περιορισμός:

Τελικά, το TSP αντιμετωπίζεται ως πρόβλημα μικτού προγραμματισμού.

5 Μοντελοποίηση – Επίλυση σε Python

Η μοντελοποίηση και επίλυση του προβλήματος έγινε σε Python.

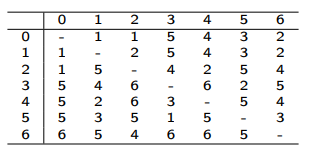
Συγκεκριμένα η μοντελοποίηση έγινε με το πακέτο [Pyomo](http://www.pyomo.org/) και η επίλυση του μοντέλου με τον IBM CPLEX solver.

Για έλεγχο του μοντέλου μου, χρησιμοποίησα datasets από την βιβλιοθήκη [TSPLIB](http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/), που περιέχει σετ λυμένων προβλημάτων TSP με διαφορετικό πλήθος κόμβων. Τα δεδομένα είτε περιέχουν πίνακες αποστάσεων για της κόμβους, είτε συντεταγμένες των κόμβων.

Η διαχείριση των αρχείων .tsp έγινε με την βιβλιοθήκη [tsplib95](https://pypi.org/project/tsplib95/).

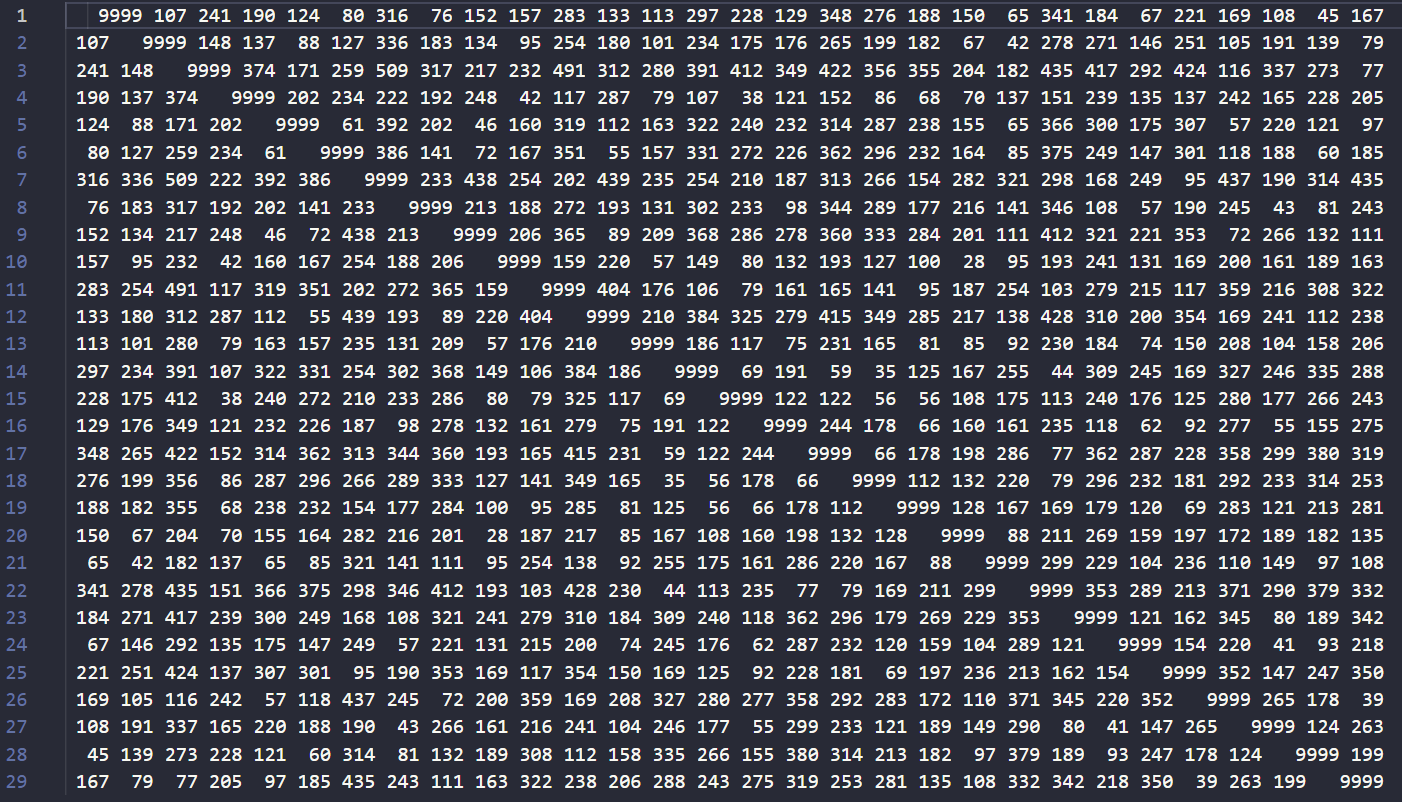
Σε περίπτωση που έχω τις συντεταγμένες, ο πίνακας αποστάσεων υπολογίζεται από την ευκλείδεια απόσταση μεταξύ των κόμβων.

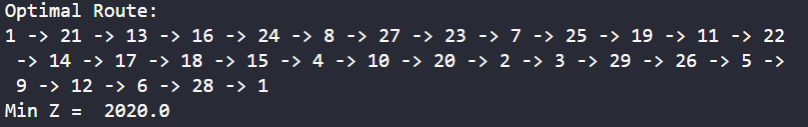
Ενδεικτικά παρουσιάζεται η λύση του machine scheduling problem των διαφανειών του μαθήματος με πίνακα αποστάσεων





Και η επίλυση της προβλήματος 29 κόμβων με πίνακα αποστάσεων





Παρακάτω παρουσιάζεται ο απαιτούμενος χρόνος επίλυσης της προβλήματος TSP συναρτήσει του πλήθους των κόμβων επίσκεψης.

|  |  |
| --- | --- |
| Number of Cities | Time to solve [s] |
| 5 | 0.02 |
| 6 | 0.02 |
| 15 | 0.03 |
| 17 | 0.09 |
| 25 | 0.08 |
| 26 | 0.45 |
| 29 | 0.63 |
| 48 | 117 |
| 52 | 2.78 |
| 70 | 212 |
| 76 | 3.5 |
| 100 | 270 |
| 120 | 379 |

Εικόνα 5: Χρόνος επίλυσης TSP με ακέραιο προγραμματισμό

Παρατηρούμε πως η επίλυση προβλημάτων TSP με βέλτιστο τρόπο είναι μια χρονοβόρα και υπολογιστικά απαιτητική διαδικασία. Ακόμα και για μικρό πλήθος κόμβων της τάξης του 102 απαιτούνται αρκετά λεπτά σε έναν απλό υπολογιστή. Μάλιστα για ακόμα μεγαλύτερα προβλήματα της τάξης των 200 κόμβων, η επίλυση δεν ήταν εφικτή, αφού το δέντρο της μεθόδου branch&bound δεν ήταν διαχειρίσιμο από το σύστημα.

Παρατηρούμε πως υπήρχαν προβλήματα TSP που λόγω της μορφής του πίνακα αποστάσεων, λύνονταν σε ελάχιστο χρόνο, αφού οι διαδικασίες της προεργασίας από τον επιλυτή, καθώς και τα cutoffs στο δέντρο ήταν ιδιαίτερα αποτελεσματικές.

Παρόλα αυτά, η πλειοψηφία των προβλημάτων είχε υψηλή χρονική και χωρική πολυπλοκότητα, γεγονός που καθιστά αναγκαία τη χρήση των παραπάνω προσεγγιστικών μεθόδων.

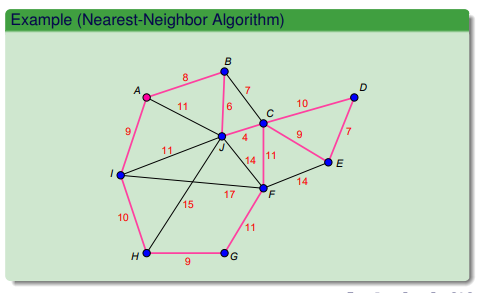
6 Λύση TSP με αλγόριθμο Nearest Neighbor

Ο αλγόριθμος NN είναι μια άπληστη (greedy) προσέγγιση στην επίλυση του προβλήματος TSP. Επιλέγει πάντα την διαδρομή ελάχιστου κόστους από τον τρέχον κόμβο προς επόμενο που δεν βρίσκεται ήδη στην διαδρομή.

Η λύση που δίνει ο αλγόριθμος εξαρτάται από τον κόμβο αφετηρίας. Για βελτίωση του, λοιπόν, μπορούμε να χρησιμοποιήσουμε ως αφετηρία όλες τις πόλεις του γράφου (ή μερικές με τυχαίο τρόπο αν έχουμε υψηλό πλήθος κόμβων) και να επιλέξουμε ως τελική λύση την διαδρομή με το ελάχιστο κόστος.

Προφανώς διατάσσουμε την λύση με τέτοιο τρόπο ώστε η αφετηρία και ο τερματισμός να είναι πάντα η πόλη 1 (η αφετηρία του πωλητή).

Είναι σημαντικό να τονίσουμε πως ο αλγόριθμος ΝΝ δεν δίνει αναγκαστικά τη βέλτιστη λύση του TSP αλλά όπως φαίνεται από τα αποτελέσματα παρακάτω, η λύση είναι αρκετά κοντά στη βέλτιστη και δίνεται σε ελάχιστο χρόνο αφού δεν πραγματοποιούνται αναδρομικές διαδικασίες κατά την επίλυση.



Εικόνα 6: Εφαρμογή αλγορίθμου ΝΝ με αφετηρία την πόλη Α

Παρακάτω συγκρίνεται ο αλγόριθμος ΝΝ με τη βέλτιστη λύση των προβλημάτων

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Number of Cities | Integer Programming  time [s] | Optimal Solution | NN Solution | NN Time [s] | Error % |
| 5 | 0.02 | 299 | 327 | 0.0076 | 9.4% |
| 6 | 0.02 | 17 | 17 | 0.0008 | 0.0% |
| 15 | 0.03 | 291 | 291 | 0.0033 | 0.0% |
| 17 | 0.09 | 2085 | 2178 | 0.017 | 4.5% |
| 25 | 0.08 | 849 | 993 | 0.018 | 17.0% |
| 26 | 0.45 | 937 | 965 | 0.012 | 3.0% |
| 29 | 0.63 | 2020 | 2134 | 0.017 | 5.6% |
| 48 | 117 | 33551 | 37928 | 0.067 | 13.0% |
| 52 | 2.78 | 7542 | 8181 | 0.11 | 8.5% |
| 70 | 212 | 675 | 796 | 0.2 | 17.9% |
| 76 | 3.5 | 538 | 608 | 0.25 | 13.0% |
| 100 | 270 | 21282 | 24698 | 0.57 | 16.1% |
| 120 | 379 | 6942 | 8438 | 1 | 21.5% |

Εικόνα 7: Σύγκριση χρόνου επίλυσης ακέραιου προγραμματισμού – ΝΝ

Παρατηρούμε πως ο χρόνος που απαιτείται για τη λειτουργία του αλγορίθμου ΝΝ είναι αμελητέος μπροστά στη μέθοδο ακεραίου προγραμματισμού.

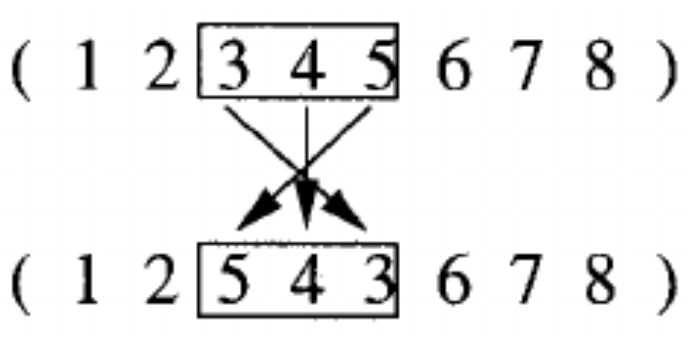
Η καταλληλόλητα του εξαρτάται από το αν το σφάλμα στην αντικειμενική συνάρτηση (10% με 20%) θεωρείται αποδεκτό στα πλαίσια του προβλήματος.

Η αδυναμία του αλγορίθμου NN έγκειται στο ότι είναι της ντετερμινιστικός αλγόριθμος χωρίς περιθώρια βελτίωσης της τελικής λύσης.

7 Βελτίωση λύσης με γενετικό αλγόριθμο

Αρχικά εφαρμόζουμε τον αλγόριθμο ΝΝ για να λάβουμε μια σχετικά καλή λύση. Έπειτα δημιουργούμε ένα πληθυσμό λύσεων (που αρχικά είναι ίδιες με την αρχική) και πραγματοποιούμε μεταλλάξεις για ένα συγκεκριμένο αριθμό επαναλήψεων που ορίζουμε εμείς.

Ως μετάλλαξη ορίζουμε την εναλλαγή δύο ενδιάμεσων πόλεων με τυχαίο τρόπο, ώστε με πολλαπλές επαναλήψεις να προκύψει πληθώρα διαφορετικών λύσεων που διαφέρουν από την αρχική.



Σε κάθε επανάληψη κρατάμε αμετάβλητο ένα ποσοστό (π.χ. 10%) των καλύτερων λύσεων (με βάση την τιμή της αντικειμενικής συνάρτησης) και μεταλλάσσουμε τις ίδιες καλές λύσεις, καθώς και τις υπόλοιπες.

Στο τέλος της κάθε επανάληψης κρατάμε τον πληθυσμό σταθερό, απορρίπτοντας ένα ποσοστό των χειρότερων λύσεων.

Έτσι, στο τέλος του αλγορίθμου έχουμε κρατήσει μεταλλάξεις που οδηγούν διαδοχικά σε μια καλύτερη λύση, χωρίς να γνωρίζουμε αν αυτή είναι η βέλτιστη.

Ο γενετικός αλγόριθμος είναι μη – ντετερμινιστικός, άρα κάθε φορά που χρησιμοποιείται μπορεί τελικά να δώσει διαφορετική λύση στο ίδιο πρόβλημα TSP.

Στα πλαίσια της εργασίας χρησιμοποίησα πληθυσμό 10n και πλήθος επαναλήψεων 50n, όπου n το πλήθος των κόμβων του TSP.

Παρακάτω φαίνονται τα αποτελέσματα του αλγορίθμου.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Number of Cities | Optimal Solution | NN Solution | Error % | Genetic Solution | Genetic Error % |
| 5 | 299 | 327 | 9.4% | 299 | 0.0% |
| 6 | 17 | 17 | 0.0% | 17 | 0.0% |
| 15 | 291 | 291 | 0.0% | 291 | 0.0% |
| 17 | 2085 | 2178 | 4.5% | 2085 | 0.0% |
| 25 | 849 | 993 | 17.0% | 941 | 10.8% |
| 26 | 937 | 965 | 3.0% | 961 | 2.6% |
| 29 | 2020 | 2134 | 5.6% | 2035 | 0.7% |
| 48 | 33551 | 37928 | 13.0% | 37255 | 11.0% |
| 52 | 7542 | 8181 | 8.5% | 8040 | 6.6% |
| 70 | 675 | 796 | 17.9% | 739 | 9.5% |
| 76 | 538 | 608 | 13.0% | 600 | 11.5% |
| 100 | 21282 | 24698 | 16.1% | 22597 | 6.2% |
| 120 | 6942 | 8438 | 21.5% | 8163 | 17.6% |

Εικόνα 8: Τελική σύγκριση αλγορίθμων

Σε κάποιες περιπτώσεις είχαμε ικανοποιητική σύγκλιση προς την βέλτιστη λύση (π.χ για 25, 70, 100 κόμβους) και σε άλλες περιπτώσεις η βελτίωση ήταν μικρή.   
Πιθανώς με περισσότερες επαναλήψεις το αποτέλεσμα να ήταν διαφορετικό.

8 Συμπεράσματα

Η εύρεση της βέλτιστης λύσης προβλημάτων πλανόδιου πωλητή επιτυγχάνεται μόνο με μεθόδους γραμμικού προγραμματισμού ή δυναμικού προγραμματισμού, οι οποίες όμως υστερούν σε ταχύτητα και απαιτήσεις μνήμης. Πρακτικά, η επίλυση προβλημάτων TSP με πολλούς κόμβους με τους παραπάνω τρόπους είναι αδύνατη για έναν απλό υπολογιστή και γενικότερα δεν συμφέρει χρονικά και οικονομικά σε βιομηχανικές – επαγγελματικές εφαρμογές.

Οι εναλλακτικές μέθοδοι που παρουσιάστηκαν παραπάνω είναι μια καλή αφετηρία για γρήγορη και σχετικά ικανοποιητική εύρεση λύσεων σε τέτοια προβλήματα, με απόκλιση από την βέλτιστη λύση περίπου 20% και σημαντική βελτίωση στην ταχύτητα εκτέλεσης.

9 Πιθανές βελτιώσεις – Εναλλακτικοί τρόποι λύσης

* Εναλλακτική μοντελοποίηση – αντιμετώπιση των εσωτερικών κυκλικών διαδρομών:

Οι περιορισμοί με χρήση βοηθητικών μεταβλητών χρησιμοποιήθηκαν επειδή είναι εύκολοι στην υλοποίηση, αφού ορίζονται μία φορά στην έναρξη του προβλήματος, αλλά εισάγουν πολυπλοκότητα κατά την επίλυση. Θα ήταν πιο αποδοτική η κλασσική μέθοδος αντιμετώπισης των εσωτερικών κυκλικών διαδρομών, με εισαγωγή περιορισμών μόνο όταν προκύψουν τέτοιες διαδρομές κατά την επίλυση του ακέραιου προβλήματος.

* Εφαρμογή αλγορίθμων τοπικής αναζήτησης με clustering για εύρεση προσεγγιστικών λύσεων με αποδοτικό τρόπο.
* Εφαρμογή self – organizing maps στα προβλήματα 2D – plane TSP για γρήγορη και με μικρό σφάλμα λύση. Η επίλυση με self – organizing map είναι από τις προτιμότερες μεθόδους στον ακαδημαϊκό – ερευνητικό κύκλο.

10 Βιβλιογραφία

[1] Σ. Δασκαλάκη, Τμήμα ΗΜ&ΤΥ Πανεπιστημίου Πάτρας – “Διαφάνειες μαθήματος Γραμμικής & Συνδυαστικής βελτιστοποίησης“

[2] Dr. Leena Jain, Mr. Amit Bhanot, “[Traveling Salesman Problem: A Case Study](https://www.researchgate.net/publication/324987377_Traveling_Salesman_Problem_A_Case_Study)”

[3] Rajesh Matai, Surya Prakash Singh and Murari Lal Mittal, “[Traveling Salesman Problem: An Overview of Applications, Formulations, and Solution Approaches](https://www.intechopen.com/chapters/12736)”

[4] yuan yuan, Diego Cattaruzza, Maxime Ogier, Frédéric Semet, “[A note on the lifted Miller-Tucker-Zemlin subtour elimination constraints for routing problems with time windows](https://hal.archives-ouvertes.fr/hal-02947086/document)”

[5] Diego Olivier Fernandez, “[Traveling Salesman Problem (TSP) with Miller-Tucker-Zemlin (MTZ) in CPLEX/OPL](https://co-enzyme.fr/blog/traveling-salesman-problem-tsp-in-cplex-opl-with-miller-tucker-zemlin-mtz-formulation/)”

[6] Claudemir Woche V.C, “[Modeling and solving the Traveling salesman problem with Python and Pyomo](http://www.opl.ufc.br/post/tsp/)”

[7] Robb T. Koether, “[The Traveling Salesman Problem Nearest-Neighbor Algorithm](http://people.hsc.edu/faculty-staff/robbk/Math111/Lectures/Fall%202016/Lecture%2033%20-%20The%20Nearest-Neighbor%20Algorithm.pdf)”

[8] Eric Stoltz, “[*Evolution* of a salesman: A complete genetic algorithm tutorial for Python](https://towardsdatascience.com/evolution-of-a-salesman-a-complete-genetic-algorithm-tutorial-for-python-6fe5d2b3ca35)*”*

[9] Gustavo Erick Anaya Fuentes, Eva Selene Hernández Gress, Juan Carlos Seck Tuoh Mora, Joselito Medina Marín, “[Solution to travelling salesman problem by clusters and a modified multi-restart iterated local search metaheuristic](https://doi.org/10.1371/journal.pone.0201868)”

# [10] Diego Vicente “[Using Self-Organizing Maps to solve the Traveling Salesman Problem](https://diego.codes/post/som-tsp/)”