Traveling Salesman Problem

Τελική εργασία του μαθήματος 'Γραμμική & Συνδυαστική βελτιστοποίηση'

Στάικος Θεόδωρος 1066578

<u>1 Περιγραφή του προβλήματος</u>

Το 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.

2. 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.

3. 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 $\frac{(n-1)!}{2}$, 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

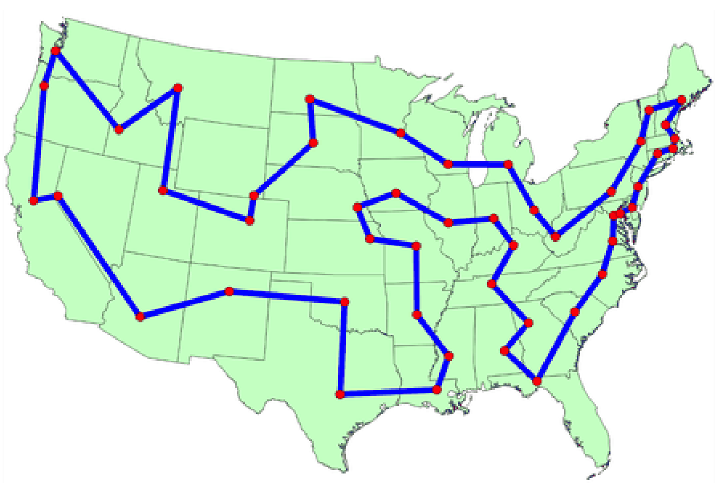
Γενικά, ως TSP μοντελοποιούνται πληθώρα προβλημάτων και εφαρμογών στη βιομηχανία.

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.

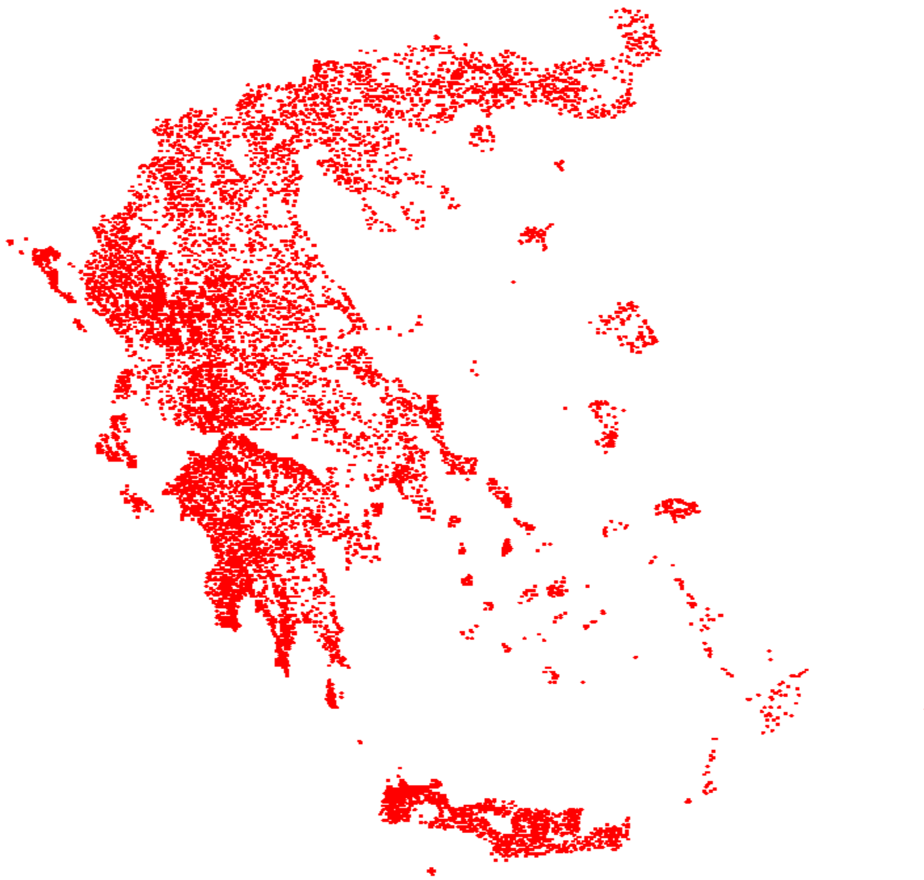
Η μοντελοποίηση προβλημάτων ως TSP problems συμβαίνει και στη βιομηχανία των ημιαγωγών για βέλτιστη κατασκευή μασκών πυριτίου, για ελάχιστη διαδρομή της κεφαλής εργαλείου κατά την διάτρηση μεταξύ επιπέδων πλακών πυριτίου και βέλτιστη καλωδίωση των παραπάνω πλακών ώστε να αποφεύγεται η περιττή χρήση υλικού.

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.

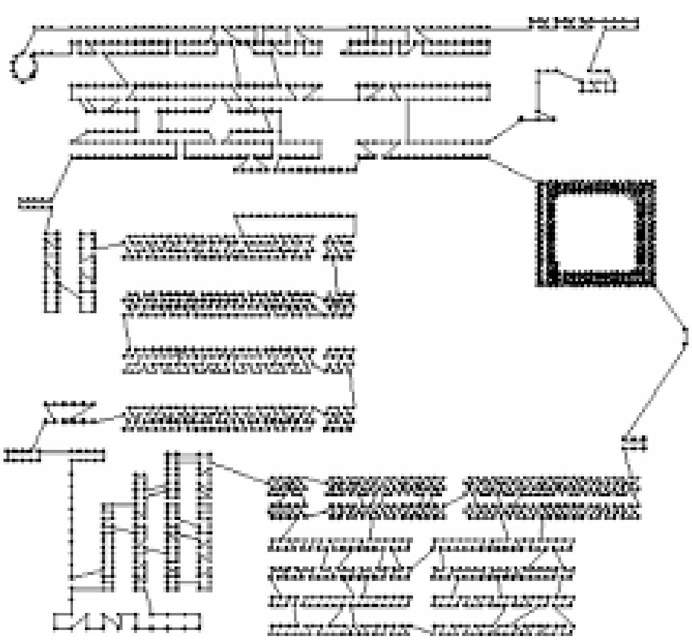
Να σημειωθεί πως τα προβλήματα TSP ταυτίζονται με τα προβλήματα προγραμματισμού μηχανής (machine scheduling programs) και η μοντελοποίηση και επίλυσή της είναι ακριβώς ίδια.



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 $\frac{(n-1)!}{2}$, άρα είναι προφανές πως η brute force επίλυση του προβλήματος δεν είναι κατάλληλη για υψηλό αριθμό κόμβων.

2. 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.

3. 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 $O(n^2)$ 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.

4. 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, 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 x_{ij} 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 $c_{ij}x_{ij}$ όπου c_{ij} το κόστος της διαδρομής από πόλη i σε πόλη j.

Άρα έχω πρόβλημα ελαχιστοποίησης

$$\min Z = \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} c_{ij}x_{ij}$$

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

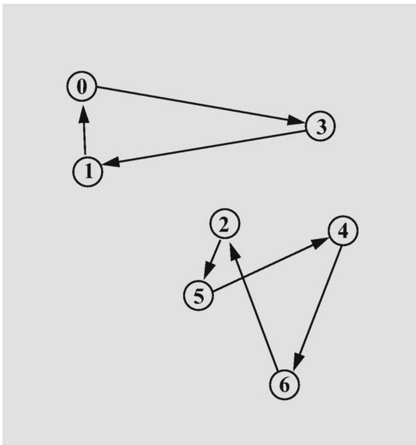
Περιορισμός 1^{ος}: Άφιξη σε κάθε πόλη ακριβώς μία φορά

$$\sum_{i=1, i\neq j}^{n} x_{ij} = 1 \; για \; j = 1, .., n$$

Περιορισμός 2^{ος}: Αναχώρηση από κάθε πόλη ακριβώς μία φορά

$$\sum_{j=1, j\neq i}^{n} x_{ij} = 1 \; για \; i = 1, .., n$$

Ο $3^{ος}$ 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 $3^{ος}$ περιορισμός εκφράζεται ως:

$$\sum_{i,j \in S} x_{ij} \leq |S| \; για\; όλα\; τα\; γνήσια\; μη-κενά\; υποσύνολα\; S \subset \{1,..,n\}$$

Με χρήση βοηθητικών μεταβλητών $u_i \in Z^{+}, i = 1,..,n$ μπορώ να εξαλείψω τις κυκλικές διαδρομές ως εξής:

Αν $x_{ij} = 1$ τότε $u_i + 1 = u_j$ (Ο κόμβος i υστερεί κατά 1 θέση του κόμβου j).
Ουσιαστικά, οι μεταβλητές u εκφράζουν την σειρά επίσκεψης των αντίστοιχων κόμβων σε μία διαδρομή.
Για παράδειγμα, αν έχω τη διαδρομή 1 -> 3 -> 2 τότε $u_1 = 1, u_2 = 3, u_3 = 2$.

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.

$$u_i - u_j + n x_{ij} \leq n-1 \; με \; 2 \leq i,j \leq n \; και \; i \neq j$$

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]

Και ο $4^{ος}$ περιορισμός:

$$x_{ij} \in \{0,1\} \; με \; i,j = 1,..,n$$

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 και η επίλυση του μοντέλου με τον IBM CPLEX solver.

Για έλεγχο του μοντέλου μου, χρησιμοποίησα datasets από την βιβλιοθήκη TSPLIB, 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.

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

	0	1	2	3	4	5	6
0	-	1	1	5	4	3	2
1	1	-	2	5	4	3	2
2	1	5	-	4	2	5	4
3	5	4	6	-	6	2	5
4	5	2	6	3	-	5	4
5	5	3	5	1	5	-	3
6	6	5	4	6	6	5	-

```
Optimal Route:
1 -> 3 -> 5 -> 4 -> 6 -> 7 -> 2 -> 1
Min Z =  17.0
```

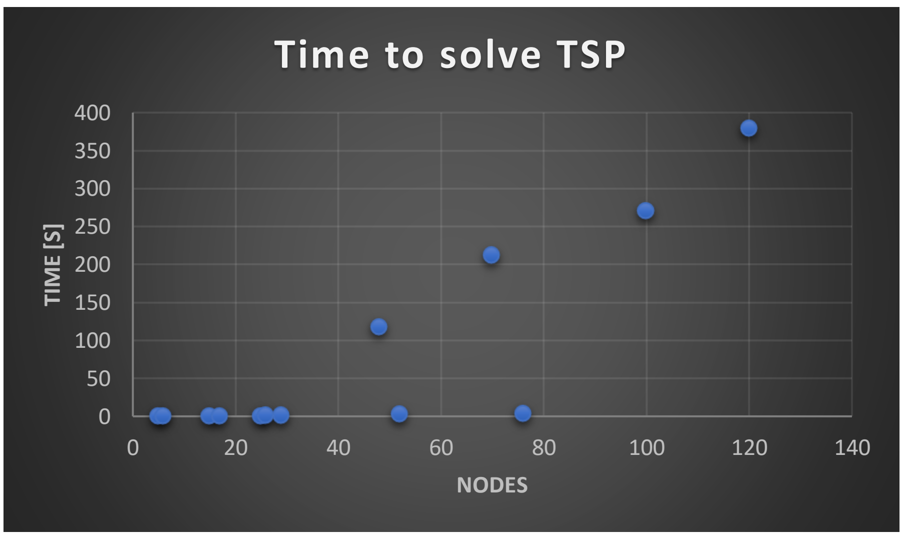
Και η επίλυση της προβλήματος 29 κόμβων με πίνακα αποστάσεων

```
 9999 107 241 190 124  80 316  76 152 157 283 133 113 297 228 129 348 276 188 150  65 341 184  67 221 169 108  45 167
107  9999 148 137  88 127 336 183 134  95 254 180 101 234 175 176 265 199 182  67  42 278 271 146 251 105 191 139  79
241 148  9999 374 171 259 509 317 217 232 491 312 280 391 412 349 422 356 355 204 182 435 417 292 424 116 337 273  77
190 137 374  9999 202 234 222 192 248  42 117 287  79 107  38 121 152  86  68  70 137 151 239 135 137 242 165 228 205
124  88 171 202  9999  61 392 202  46 160 319 112 163 322 240 232 314 287 238 155  65 366 300 175 307  57 220 121  97
 80 127 259 234  61  9999 386 141  72 167 351  55 157 331 272 226 362 296 232 164  85 375 249 147 301 118 188  60 185
316 336 509 222 392 386  9999 233 438 254 202 439 235 254 210 187 313 266 154 282 321 298 168 249  95 437 190 314 435
 76 183 317 192 202 141 233  9999 213 188 272 193 131 302 233  98 344 289 177 216 141 346 108  57 190 245  43  81 243
152 134 217 248  46  72 438 213  9999 206 365  89 209 368 286 278 360 333 284 201 111 412 321 221 353  72 266 132 111
157  95 232  42 160 167 254 188 206  9999 159 220  57 149  80 132 193 127 100  28  95 193 241 131 169 200 161 189 163
283 254 491 117 319 351 202 272 365 159  9999 404 176 106  79 161 165 141  95 187 254 103 279 215 117 359 216 308 322
133 180 312 287 112  55 439 193  89 220 404  9999 210 384 325 279 415 349 285 217 138 428 310 200 354 169 241 112 238
113 101 280  79 163 157 235 131 209  57 176 210  9999 186 117  75 231 165  81  85  92 230 184  74 150 208 104 158 206
297 234 391 107 322 331 254 302 368 149 106 384 186  9999  69 191  59  35 125 167 255  44 309 245 169 327 246 335 288
228 175 412  38 240 272 210 233 286  80  79 325 117  69  9999 122 122  56  56 108 175 113 240 176 125 280 177 266 243
129 176 349 121 232 226 187  98 278 132 161 279  75 191 122  9999 244 178  66 160 161 235 118  62  92 277  55 155 275
348 265 422 152 314 362 313 344 360 193 165 415 231  59 122 244  9999  66 178 198 286  77 362 287 228 358 299 380 319
276 199 356  86 287 296 266 289 333 127 141 349 165  35  56 178  66  9999 112 132 220  79 296 232 181 292 233 314 253
188 182 355  68 238 232 154 177 284 100  95 285  81 125  56  66 178 112  9999 128 167 169 179 120  69 283 121 213 281
150  67 204  70 155 164 282 216 201  28 187 217  85 167 108 160 198 132 128  9999  88 211 269 159 197 172 189 182 135
 65  42 182 137  65  85 321 141 111  95 254 138  92 255 175 161 286 220 167  88  9999 299 229 104 236 110 149  97 108
341 278 435 151 366 375 298 346 412 193 103 428 230  44 113 235  77  79 169 211 299  9999 353 289 213 371 290 379 332
184 271 417 239 300 249 168 108 321 241 279 310 184 309 240 118 362 296 179 269 229 353  9999 121 162 345  80 189 342
 67 146 292 135 175 147 249  57 221 131 215 200  74 245 176  62 287 232 120 159 104 289 121  9999 154 220  41  93 218
221 251 424 137 307 301  95 190 353 169 117 354 150 169 125  92 228 181  69 197 236 213 162 154  9999 352 147 247 350
169 105 116 242  57 118 437 245  72 200 359 169 208 327 280 277 358 292 283 172 110 371 345 220 352  9999 265 178  39
108 191 337 165 220 188 190  43 266 161 216 241 104 246 177  55 299 233 121 189 149 290  80  41 147 265  9999 124 263
 45 139 273 228 121  60 314  81 132 189 308 112 158 335 266 155 380 314 213 182  97 379 189  93 247 178 124  9999 199
167  79  77 205  97 185 435 243 111 163 322 238 206 288 243 275 319 253 281 135 108 332 342 218 350  39 263 199  9999
```

```
Optimal Route:
1 -> 21 -> 13 -> 16 -> 24 -> 8 -> 27 -> 23 -> 7 -> 25 -> 19 -> 11 -> 22
 -> 14 -> 17 -> 18 -> 15 -> 4 -> 10 -> 20 -> 2 -> 3 -> 29 -> 26 -> 5 ->
 9 -> 12 -> 6 -> 28 -> 1
Min Z =  2020.0
```

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 10^2 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.

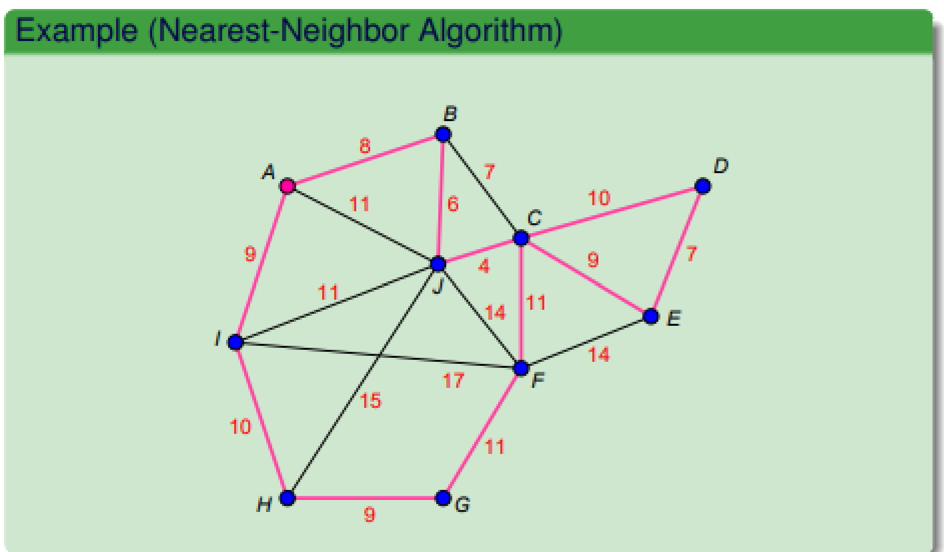
6 Λύση TSP με αλγόριθμο Nearest Neighbor

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).

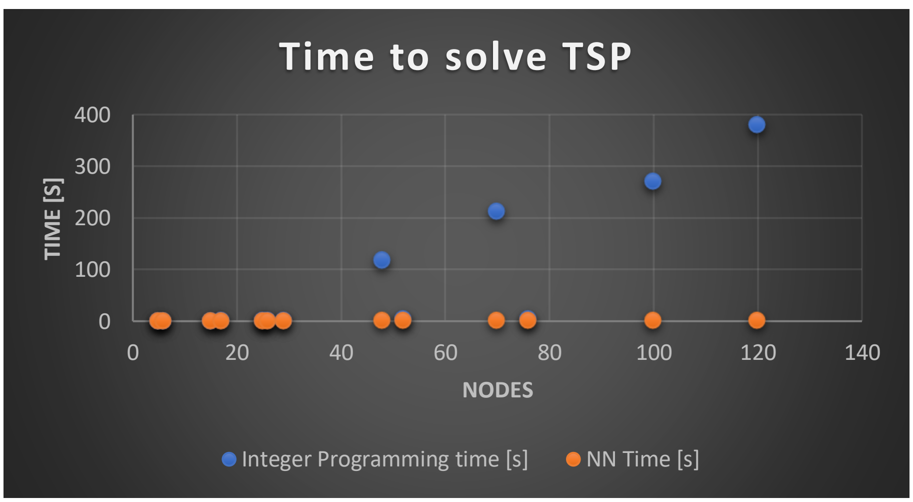
Είναι σημαντικό να τονίσουμε πως ο αλγόριθμος NN δεν δίνει αναγκαστικά τη βέλτιστη λύση του TSP αλλά όπως φαίνεται από τα αποτελέσματα παρακάτω, η λύση είναι αρκετά κοντά στη βέλτιστη και δίνεται σε ελάχιστο χρόνο αφού δεν πραγματοποιούνται αναδρομικές διαδικασίες κατά την επίλυση.



Εικόνα 6: Εφαρμογή αλγορίθμου NN με αφετηρία την πόλη A

Παρακάτω συγκρίνεται ο αλγόριθμος NN 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 – NN

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

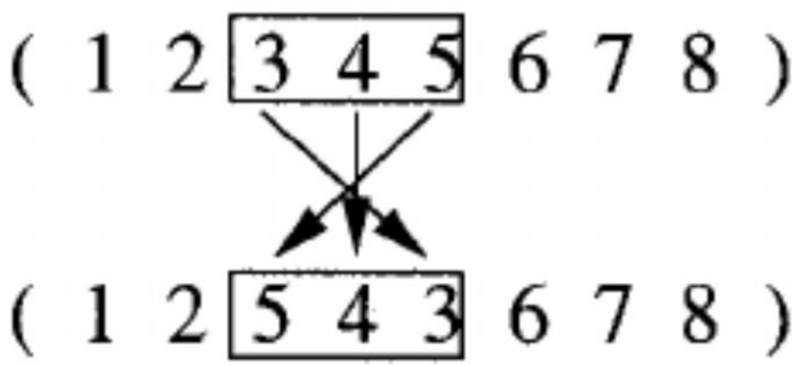
Η καταλληλόλητα του εξαρτάται από το αν το σφάλμα στην αντικειμενική συνάρτηση (10% με 20%) θεωρείται αποδεκτό στα πλαίσια του προβλήματος.

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 NN 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.

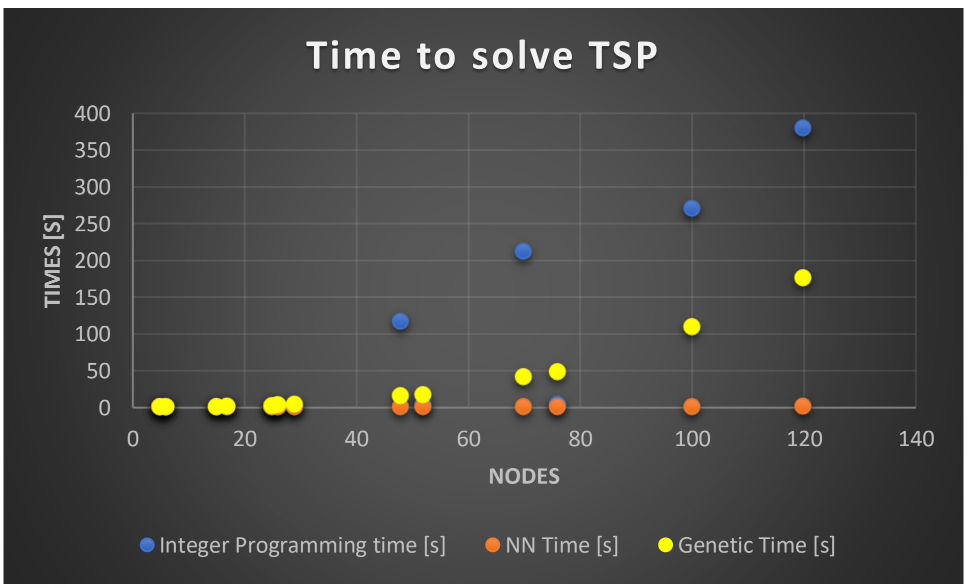
Έτσι, στο τέλος του αλγορίθμου έχουμε κρατήσει μεταλλάξεις που οδηγούν διαδοχικά σε μια καλύτερη λύση, χωρίς να γνωρίζουμε αν αυτή είναι η βέλτιστη.

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"

[3] Rajesh Matai, Surya Prakash Singh and Murari Lal Mittal, "Traveling Salesman Problem: An Overview of Applications, Formulations, and Solution Approaches"

[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"

[5] Diego Olivier Fernandez, "Traveling Salesman Problem (TSP) with Miller-Tucker-Zemlin (MTZ) in CPLEX/OPL"

[6] Claudemir Woche V.C, "Modeling and solving the Traveling salesman problem with Python and Pyomo"

[7] Robb T. Koether, "The Traveling Salesman Problem Nearest-Neighbor Algorithm"

[8] Eric Stoltz, "*Evolution* of a salesman: A complete genetic algorithm tutorial for Python"

[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"

[10] Diego Vicente "Using Self-Organizing Maps to solve the Traveling Salesman Problem"