ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΕΦΑΡΜΟΣΜΕΝΩΝ ΜΑΘΗΜΑΤΙΚΩΝ ΚΑΙ ΦΥΣΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ ΣΤΗΝ ΕΠΙΧΕΙΡΗΣΙΑΚΗ ΕΡΕΥΝΑ

ΔΥΝΑΜΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΚΑΙ ΕΦΑΡΜΟΓΕΣ

ΚΑΡΥΤΙΝΟΥ ΜΑΡΙΑ- ΝΕΦΕΛΗ

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ : ΚΟΛΕΤΣΟΣ ΙΩΑΝΝΗΣ
ΑΝΑΠΛΗΡΩΤΗΣ ΚΑΘΗΓΗΤΗΣ Ε.Μ.Π.

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ: ΚΟΚΚΙΝΗΣ ΒΑΣΙΛΕΙΟΣ, ΕΠΙΚΟΥΡΟΣ ΚΑΘΗΓΗΤΗΣ Ε.Μ.Π.
ΚΟΛΕΤΣΟΣ ΙΩΑΝΝΗΣ, ΑΝΑΠΛΗΡΩΤΗΣ ΚΑΘΗΓΗΤΗΣ Ε.Μ.Π.
ΣΤΕΦΑΝΕΑΣ ΠΕΤΡΟΣ, ΕΠΙΚΟΥΡΟΣ ΚΑΘΗΓΗΤΗΣ Ε.Μ.Π.

ΑΘΗΝΑ, ΙΑΝΟΥΑΡΙΟΣ 2021

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΕΦΑΡΜΟΣΜΕΝΩΝ ΜΑΘΗΜΑΤΙΚΩΝ ΚΑΙ ΦΥΣΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

Πτυχιακή Εργασία στην Επιχειρησιακή Έρευνα

ΔΥΝΑΜΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΚΑΙ ΕΦΑΡΜΟΓΕΣ

ΚΑΡΥΤΙΝΟΥ ΜΑΡΙΑ- ΝΕΦΕΛΗ

Επιβλέπων καθηγητής: Κολέτσος Ιωάννης
Αναπληρωτής Καθηγητής Ε.Μ.Π.

ΑΘΗΝΑ, ΙΑΝΟΥΑΡΙΟΣ 2021

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω ιδιαιτέρως τον καθηγητή μου κ. Ιωάννη Κολέτσο για την επίβλεψη της παρούσας πτυχιακής εργασίας. Οι συμβουλές του στο επίπεδο της βιβλιογραφίας, της μεθοδολογίας, της αλγοριθμικής προσέγγισης και των εφαρμογών ήταν πολύτιμες ώστε να φέρω εις πέρας την εργασία μου. Καθ' όλη τη διάρκεια ήμασταν σε συνεχή επικοινωνία, προσφέροντας μου τόσο τις γνώσεις του πάνω στο αντικείμενο της Επιχειρησιακής Έρευνας όσο και την εμπειρία του στο κομμάτι των ερευνητικών εργασιών. Έτσι κατάφερα να κατανοήσω σε μεγαλύτερο βάθος το γνωστικό αντικείμενο μελέτης αλλά και τον τρόπο εκπόνησης μίας πτυχιακής εργασίας με το σωστότερο δυνατό τρόπο.
Ακόμη, θα ήθελα να ευχαριστήσω όλους τους καθηγητές μου κατά τη διάρκεια της φοίτησης μου οι οποίοι συνέβαλλαν ο καθένας ξεχωριστά στην ολοκλήρωση των σπουδών μου, αναπόσπαστο τμήμα των οποίων αποτελεί η παρούσα πτυχιακή εργασία.

ΠΕΡΙΛΗΨΗ

Σκοπός της παρούσας πτυχιακής εργασίας είναι η μελέτη εφαρμογών του Δυναμικού Προγραμματισμού, ενός πεδίου της Επιχειρησιακής Έρευνας ιδιαιτέρως χρήσιμου και συνεχώς εξελίξιμου. Στο πλαίσιο αυτό αναλύουμε μεθόδους και τεχνικές του Δυναμικού Προγραμματισμού παραθέτοντας ενδεικτικά παραδείγματα που τις υλοποιούν. Εστιάζουμε σε εφαρμογές στις οποίες επεξεργαζόμαστε πραγματικά δεδομένα με τη χρήση της προγραμματιστικής γλώσσας Python ώστε να γίνει πλήρως κατανοητή η συμβολή του δυναμικού προγραμματισμού στην επίλυση προβλημάτων από διάφορους τομείς τόσο της επιστήμης των υπολογιστών όσο και του εμπορίου και της βιομηχανίας. Πιο αναλυτικά, στο πρώτο κεφάλαιο γίνεται μία ιστορική αναδρομή που περιλαμβάνει τις ανάγκες που αποτέλεσαν το έναυσμα για τη δημιουργία της Επιχειρησιακής Έρευνας και τους παράγοντες που συνέβαλλαν στη ραγδαία εξέλιξή της μέχρι σήμερα. Ακόμη, γίνεται μία σύντομη παρουσίαση των βημάτων για την επίλυση των προβλημάτων που αφορούν σε αυτόν τον επιστημονικό κλάδο. Στο δεύτερο κεφάλαιο γίνεται μία εκτενής εισαγωγή στο βασικό αντικείμενο μελέτης της παρούσας εργασίας, το Δυναμικό Προγραμματισμό. Παρουσιάζουμε τα βασικά χαρακτηριστικά και τη γενική μεθοδολογία επίλυσης προβλημάτων και στη συνέχεια επικεντρωνόμαστε στη βάση του μεγαλύτερου μέρους της εργασίας η οποία είναι ο Αιτιοκρατικός Δυναμικός Προγραμματισμός. Το κύριο μέρος αποτελείται από τέσσερα κεφάλαια το καθένα από τα οποία παρουσιάζει μία διαφορετική εφαρμογή του Δυναμικού Προγραμματισμού. Πιο συγκεκριμένα, το τρίτο κεφάλαιο πραγματεύεται το πρόβλημα της «απόστασης κατά Levensthein» αναλύοντας τη μεθοδολογία και παραθέτοντας μία πλήρη λύση ενός ενδεικτικού προβλήματος καθώς και μία μικρότερης έκτασης προσομοίωση μίας εφαρμογής του. Στο τέταρτο κεφάλαιο, δίνουμε ιδιαίτερη έμφαση στη μελέτη ίσως της πιο γνωστής εφαρμογής του δυναμικού προγραμματισμού που είναι το πρόβλημα της ελάχιστης διαδρομής. Δουλεύουμε δύο διαφορετικές προσεγγίσεις επίλυσης και έπειτα από δύο πρότυπα παραδείγματα παρουσιάζουμε ένα πρόβλημα βασισμένο σε πραγματικά δεδομένα. Το πρόβλημα του πλανόδιου πωλητή με το οποίο ασχολούμαστε στο πέμπτο κεφάλαιο φαίνεται να είναι ένα πρόβλημα-πρόκληση για πολλούς επιστήμονες μέχρι και σήμερα οι οποίοι διαρκώς αναζητούν νέους βελτιωμένους αλγορίθμους για την επίλυση του. Εμείς επικεντρωνόμαστε σε μία μέθοδο επίλυσης με τη χρήση ενός αλγορίθμου δυναμικού προγραμματισμού (Held & Karp). Τέλος, στο έκτο κεφάλαιο μεταβαίνουμε στην περιοχή του Πιθανοτικού Δυναμικού Προγραμματισμού και στην εισαγωγή μίας ευρύτερης περιοχής του δυναμικού προγραμματισμού που ονομάζεται Ενισχυτική Μάθηση (Reinforcement Learning). Μέσω επίλυσης μίας γνωστής εισαγωγικής εφαρμογής, αυτής του προβλήματος της Παγωμένης Λίμνης, παρουσιάζουμε ένα σύντομο αλλά κατατοπιστικό πρώτο δείγμα της συνεχούς εξέλιξης αυτού του τομέα καθώς οι προοπτικές που φαίνονται να διαγράφονται πίσω από τις εφαρμογές του δυναμικού προγραμματισμού και της ενισχυτικής μάθησης είναι ικανές να αλλάξουν την εξελικτική πορεία των επιστημών και κατ’ επέκταση τον τρόπο που ζούμε.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1

Επιχειρησιακή Έρευνα

1.1 ΕΙΣΑΓΩΓΗ- ΙΣΤΟΡΙΚΗ ΑΝΑΔΡΟΜΗ

Η Επιχειρησιακή Έρευνα (Ε.Ε.), μέσα από μια αναδρομή μερικών αιώνων, θα λέγαμε ότι είναι μια σχετικά νέα επιστήμη. Οι ολοένα αυξανόμενες ανάγκες των επιχειρήσεων να διαχειριστούν μικρά ή μεγαλύτερα προβλήματα έθεσε τα θεμέλια ώστε να δημιουργηθεί ένας κλάδος ο οποίος στοχεύει στην εύρεση των καλύτερων δυνατών λύσεων αυτών των προβλημάτων προσεγγίζοντάς τα επιστημονικά [1]. Η σωστή κατανομή των ευθυνών, ο καταμερισμός των εργασιών ακόμα και ο διαχωρισμός των στόχων μιας επιχείρησης είναι μερικά από τα προβλήματα που πρωτοεμφανίστηκαν καθώς το μέγεθος των επιχειρήσεων άρχισε να διογκώνεται. Παράλληλα, οι απαιτήσεις λόγω αύξησης της εξειδίκευσης φάνηκαν από τα πρώτα κιόλας στάδια της βιομηχανικής επανάστασης (19ος αιώνας) και έπρεπε να αντιμετωπιστούν άμεσα.

Πατέρας της Επιχειρησιακής Έρευνας θεωρείται ο Βρετανός μαθηματικός Charles Babbage (1791-1871). Προχώρησε σε μία έρευνα για το κόστος μεταφοράς και το κόστος για την ταξινόμηση της αλληλογραφίας που είχε ως αποτέλεσμα τη δημιουργία του λεγόμενου «Ταχυδρομείου της πέννας» (Uniform Penny Post) το 1840 [2].

Ωστόσο, οι διάφορες ανάγκες λήψης αποφάσεων που προέκυψαν από τις αρχές του Β' Παγκοσμίου Πολέμου (1940) ήταν αυτές που οδήγησαν την Επιχειρησιακή Έρευνα να καθιερωθεί πλέον ως ένας νέος, ξεχωριστός επιστημονικός κλάδος. Την αρχή {[2], [63], [64]} έκανε η Μεγάλη Βρετανία η οποία συνέθεσε μια ομάδα από εξιδεικευμένους επιστήμονες διαφόρων κλάδων όπως μηχανικούς και μαθηματικούς προκειμένου να μελετήσουν την ενσωμάτωση της τεχνολογίας των ραντάρ για τον εντοπισμό των εχθρικών αεροσκαφών και συνεπώς την αποτελεσματικότερη εναέρια άμυνά της. Στη συνέχεια, ακολούθησαν έρευνες σχετικές με στρατιωτικά προβλήματα που προκύπταν τόσο στο πεζικό όσο και στις θαλάσσιες δυνάμεις της. Μελετήθηκαν ερωτήματα που αφορούσαν τη θαλάσσια κυριαρχία, όπως το κατάλληλο μέγεθος των νηοπομπών[1] (convoys) και χρησιμοποιήθηκαν τεχνικές προκειμένου να βελτιωθούν οι επιθετικές κινήσεις των Βρετανών.

Η είσοδος των Αμερικανών στον Β' Παγκόσμιο Πόλεμο έγινε λίγα χρόνια αργότερα (1942). Η έως τότε ανάπτυξη της Επιχειρησιακής Έρευνας από τους Βρετανούς είχε σημαντικό αντίκτυπο στις στρατιωτικές δραστηριότητες του Αμερικανικού Στρατού. Υιοθέτησαν αντίστοιχες τεχνικές για τη χρήση των ραντάρ και ανέπτυξαν δικά τους μοντέλα ώστε να ελέγξουν τις τακτικές που ακολουθούνταν για τη χρήση και εκμετάλλευση των όπλων και των πυρομαχικών τους. Η επιχειρησιακή έρευνα φάνηκε ιδιαιτέρως χρήσιμη στην αμερικανική πολεμική αεροπορία και το πολεμικό ναυτικό με σημαντικά αποτελέσματα όπως η βελτιστοποίηση της αρχικής προσπάθειας των Αμερικανών για προσδιορισμό του μεγέθους των «convoys». Μάλιστα, η αύξηση του μεγέθους τους αποδείχθηκε καθοριστική για τη νίκη στη μάχη του Ατλαντικού. Σημαντικό γεγονός, επίσης, τον Οκτώβριο του 1942 αποτέλεσε η συνεργασία ομάδων των δύο δυνάμεων για τη βελτιστοποίηση της προσπάθειας τους να αυξήσουν την ακρίβεια στη θέση που τοποθετούνταν οι βόμβες με σκοπό την

[1] Ομάδα φορτηγών πλοίων που συμπλέουν οργανωμένα υπό την προστασία πολεμικών πλοίων

αποτελεσματικότερη πλήξη των αντίπαλων στόχων. Συνολικά, οι βρετανικές και αμερικανικές ειδικές δυνάμεις κατάφεραν να λύσουν πολλά προβλήματα και να προτείνουν στρατηγικές και τακτικές κινήσεις σε διάφορες στρατιωτικές επιχειρήσεις κατά τη διάρκεια του Β' Παγκοσμίου Πολέμου. Αυτή η προσέγγιση μάλιστα ήταν η αιτία για την ονομασία «επιχειρησιακή έρευνα» και πιο συγκεκριμένα "Operational Research" στη Βρετανία και "Operations Research" στην Αμερική [1].

Όταν ο πόλεμος τελείωσε ήταν πλέον σαφές ότι οι δυνατότητες της Επιχειρησιακής Έρευνας μπορούσαν να επεκταθούν και σε άλλους τομείς όπως αυτοί των επιχειρήσεων, ακόμα και της κυβέρνησης για τη λύση κυρίως οργανωτικών και διοικητικών προβλημάτων. Στη συνέχεια ακολούθησε μία ραγδαία ανάπτυξη του κλάδου της Επιχειρησιακής Έρευνας που συνεχίζει μέχρι και σήμερα. Πρωταρχικό λόγο για αυτή την αξιοσημείωτη εξέλιξη αποτελεί η άμεση εφαρμογή και η λύση πολύπλοκων προβλημάτων που έδωσε κίνητρο σε πολλούς επιστήμονες να ασχοληθούν σε βάθος με αυτή τη νέα επιστήμη, να βελτιώσουν και να ανακαλύψουν νέες τεχνικές. Ήδη από τα μέσα της δεκαετίας του 1950 άρχισαν να εφαρμόζονται μέθοδοι όπως ο γραμμικός προγραμματισμός, ο δυναμικός προγραμματισμός, η θεωρία αποθεμάτων και η θεωρία ουρών αναμονής. Χαρακτηριστικό παράδειγμα αποτελεί η «Μέθοδος Simplex» για τη λύση προβλημάτων γραμμικού προγραμματισμού που αναπτύχθηκε από τον G. Dantzig (1947). Ακόμη, η ανάπτυξη των ηλεκτρονικών υπολογιστών και η πρόοδος της τεχνολογίας δημιούργησαν τις προϋποθέσεις για την πραγματοποίηση ερευνών που πριν φάνταζαν αδύνατες, είτε λόγω του μεγέθους των αριθμητικών υπολογισμών με το χέρι είτε λόγω της χρονοβόρας διαδικασίας. Με τη χρήση, λοιπόν, των υπολογιστών δημιουργήθηκαν μαθηματικά μοντέλα που χρησιμοποιήθηκαν ως βασικά εργαλεία για την επίλυση τέτοιων προβλημάτων και μειώθηκε αισθητά ο χρόνος κάθε έρευνας με στόχο τη δημιουργία νέων εφαρμογών της Ε.Ε.

1.2 ΤΙ ΕΙΝΑΙ ΕΠΙΧΕΙΡΗΣΙΑΚΗ ΕΡΕΥΝΑ

Η Επιχειρησιακή Έρευνα (Ε.Ε.) {[1], [2]} είναι ο κλάδος της επιστήμης των Μαθηματικών και της Πληροφορικής με αντικείμενο την εύρεση βέλτιστων λύσεων για τη λήψη αποφάσεων σε ένα περιβάλλον. Πολλές φορές αυτή η μετάφραση από τη διεθνή ονομασία Operations Research (OR) οδηγεί λανθασμένα στο συμπέρασμα ότι χρησιμοποιείται για την επίλυση προβλημάτων των επιχειρήσεων. Αντίθετα μάλιστα, αξίζει να τονίσουμε ότι η Ε.Ε. έχει πληθώρα εφαρμογών εκτός των επιχειρήσεων όπως στο στρατό, από όπου και ξεκίνησε, τη Δημόσια Υγεία, το Χρηματοοικονομικό σχεδιασμό, τις Τηλεπικοινωνίες και πολλά ακόμα. Χαρακτηριστικές εφαρμογές της είναι η διαχείριση ανθρώπινων πόρων, ο καταμερισμός του εργατικού δυναμικού στις θέσεις εργασίας, το βέλτιστο επίπεδο παραγωγής προϊόντων με στόχο το μέγιστο κέρδος ή το ελάχιστο κόστος, η ελάχιστη διαδρομή μεταφοράς προϊόντων από τα εργοστάσια στα κέντρα διανομής κ.α. Πλέον χρησιμοποιούνται ευρέως και άλλες ταυτόσημες ονομασίες όπως Διοικητική Επιστήμη (Management Science) ή συνδυαστικά Operations Research and Management Science (OR/MS) [2].

Θα ορίζαμε λοιπόν την Επιχειρησιακή Έρευνα ως την επιστήμη λήψης αποφάσεων περίπλοκων προβλημάτων με τη χρήση προηγμένων μεθόδων ανάλυσης που καταλήγουν σε βέλτιστες ή σχεδόν βέλτιστες λύσεις. Χρησιμοποιούνται διαφορετικές τεχνικές, ποσοτικές μέθοδοι και μαθηματικά μοντέλα προκειμένου να καταλήξουν στην καλύτερη δυνατή λύση του εκάστοτε προβλήματος. Σημείο κλειδί για τον παραπάνω ορισμό αποτελεί ο όρος «βέλτιστη» λύση. Αυτή η λύση μπορεί να είναι είτε μοναδική ή να υπάρχουν κι άλλες

εναλλακτικές εξίσου αποτελεσματικές που οδηγούν στην επιλογή βέλτιστης πολιτικής και στην περίπτωση αυτή επιλέγουμε μια εξ' αυτών. Γι αυτό συχνά αναφερόμαστε σε «μια» βέλτιστη λύση και όχι «στη» βέλτιστη λύση.

Τα βήματα που ακολουθούνται σε γενικό πλαίσιο για την επίλυση ενός προβλήματος Ε.Ε. είναι τα εξής {[2], [4]} :

1. Καθορισμός Προβλήματος/ Ανάλυση Συστήματος

Σε αυτό το στάδιο περιλαμβάνεται η πλήρης κατανόηση του συστήματος. Καλούμαστε να ελέγξουμε τον τυχόν διαχωρισμό σε υπό-προβλήματα, να κατανοήσουμε τους στόχους και να αναγνωρίσουμε όλες τις παραμέτρους αλλά και τους περιορισμούς που επιβάλλονται στο σύστημα. Ακόμη, συλλέγουμε όλα τα δεδομένα που είναι απαραίτητα για τη μετέπειτα μοντελοποίηση του προβλήματος.

2. Καθορισμός στόχων και σκιαγράφηση λύσεων

Ένα από τα σημαντικότερα βήματα είναι η διατύπωση των στόχων αφού από αυτούς θα εξαρτηθεί η επιτυχημένη εφαρμογή των προτεινόμενων λύσεων και ενεργειών. Χαρακτηριστικά παραδείγματα είναι η μεγιστοποίηση του κέρδους ή η ελαχιστοποίηση του κόστους μιας επιχείρησης. Παράλληλα, είναι πολύ χρήσιμο να προσδιορίζουμε εξ αρχής τις αναμενόμενες λύσεις του προβλήματος, γεγονός που υποδηλώνει ότι είμαστε σε θέση να αναγνωρίσουμε τους παράγοντες που μπορούν να επηρεάσουν το τελικό αποτέλεσμα.

3. Μαθηματική μοντελοποίηση

Στο στάδιο αυτό κατασκευάζουμε ένα απλουστευμένο μαθηματικό υπόδειγμα του πραγματικού συστήματος προκειμένου να μελετήσουμε και να λύσουμε το εκάστοτε πρόβλημα. Η διαδικασία μετατρέπει το περιγραφόμενο πρόβλημα σε μαθηματικές σχέσεις, δημιουργώντας έτσι ένα μαθηματικό μοντέλο. Το μοντέλο αυτό πρέπει να περιλαμβάνει όλες τις σημαντικές μεταβλητές που περιγράφουν το πραγματικό σύστημα και τις ορθές σχέσεις μεταξύ τους καθώς επίσης και όλους τους περιορισμούς μέσα στους οποίους το μοντέλο είναι ικανό να κινηθεί. Η διαδικασία διατύπωσης του μοντέλου χωρίζεται σε τρεις φάσεις:

I. Διατύπωση ρεαλιστικών υποθέσεων για την απλούστευση του προβλήματος.

II. Διατύπωση των μαθηματικών σχέσεων στον υπολογιστή.

III. Επικύρωση του μοντέλου ύστερα από πειραματική δοκιμή του σε ένα απλό πρόβλημα με στόχο τον έλεγχο των δύο παραπάνω βημάτων. Σε περίπτωση που δεν λάβουμε ικανοποιητικά αποτελέσματα οι φάσεις I και II ορίζονται ξανά.

4. Επίλυση μοντέλου και εύρεση λύσης

Αν το μοντέλο που κατασκευάσαμε στο στάδιο 3 είναι το σωστό, αν δηλαδή αντικατοπτρίζει το πραγματικό σύστημα, τότε η λύση ή οι λύσεις που θα λάβουμε θα είναι λύσεις του πραγματικού προβλήματος.
Η επίλυση του μοντέλου γίνεται έπειτα από επιλογή της κατάλληλης μεθόδου της Επιχειρησιακής Έρευνας αναλόγως τη φύση και την πολυπλοκότητα του μοντέλου. Μερικές μόνο από τις πιο γνωστές τεχνικές είναι ο γραμμικός προγραμματισμός που αναφέρεται σε μοντέλα με γραμμικούς περιορισμούς, ο δυναμικός προγραμματισμός που διασπά ένα γενικό πρόβλημα σε υπό-προβλήματα, ο ακέραιος προγραμματισμός που περιλαμβάνει ακέραιες μεταβλητές, ο μη-γραμμικός προγραμματισμός όπου οι συναρτήσεις των μοντέλων είναι μη-γραμμικές, τα δέντρα αποφάσεων, η θεωρία ουρών αναμονής, η θεωρία παιγνίων κ.α.

5. Ανάλυση ευαισθησίας

Με τον όρο Ανάλυση Ευαισθησίας αναφερόμαστε στη μέθοδο με την οποία ελέγχουμε την ευαισθησία/συμπεριφορά της λύσης όταν μεταβάλλουμε τις παραμέτρους του μοντέλου. Η σημαντικότητα αυτού του σταδίου έγκειται στο γεγονός ότι αυτή η ανάλυση μπορεί να διευκολύνει τον καθορισμό της στρατηγικής που θα ακολουθήσουμε.

6. Εφαρμογή της επιλεχθείσας λύσης

Συνήθως αυτό το στάδιο είναι το δυσκολότερο καθώς τα αποτελέσματα πρέπει να μετατραπούν σε οδηγίες και πρακτικές εφαρμογές στο πραγματικό σύστημα ώστε να επέλθει η βελτιστοποίηση του αληθινού προβλήματος. Είναι πολύ σημαντική η συνεχής παρακολούθηση της εξέλιξης του, αφού πολλές φορές παρουσιάζονται προβλήματα που δεν είχαν προβλεφθεί ή συμπεριληφθεί στο μοντέλο κατά τη διεξαγωγή της έρευνας.

Τέλος, ένα χαρακτηριστικό της Επιχειρησιακής Έρευνας αποτελεί η λεγόμενη «ομαδική προσέγγιση». Το φάσμα των εφαρμογών της είναι τόσο ευρύ που είναι πρακτικώς αδύνατον ένα μόνο άτομο να είναι εξιδεικευμένο σε όλα τα πιθανά προβλήματα της. Συνήθως οι ομάδες αποτελούνται από ανθρώπους διαφορετικών επιστημονικών κλάδων που είναι καλά εκπαιδευμένοι στους τομείς των μαθηματικών, της στατιστικής, των πιθανοτήτων, των υπολογιστών, της μηχανικής, των οικονομικών και των φυσικών επιστημών.
Στα επόμενα κεφάλαια θα ασχοληθούμε εκτενώς με μία κατηγορία της Επιχειρησιακής Έρευνας που ονομάζεται Δυναμικός Προγραμματισμός.

ΚΕΦΑΛΑΙΟ 2

ΔΥΝΑΜΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

2.1 ΕΙΣΑΓΩΓΗ

Ο Δυναμικός Προγραμματισμός (Dynamic Programming) [3] είναι μια αλγοριθμική μέθοδος για την επίλυση προβλημάτων βελτιστοποίησης η οποία διασπά το βασικό πρόβλημα σε μικρότερα υπό-προβλήματα δημιουργώντας έτσι μία σειρά από διαδοχικές αλληλεξαρτώμενες αποφάσεις. Στόχος είναι η εύρεση της καλύτερης δυνατής λύσης του γενικού προβλήματος η οποία εξαρτάται από τη βέλτιστη λύση των επιμέρους υπό-προβλημάτων. Συνήθως τα στάδια που ακολουθούμε προκειμένου να λύσουμε το πρόβλημα εξαρτώνται από το χρόνο εξ ου και το όνομα Δυναμικός Προγραμματισμός. Η ονομασία αυτή δόθηκε από τον Richard E. Bellman, τον μαθηματικό που εισήγαγε το Δυναμικό Προγραμματισμό [21]. Οι χρονικές μεταβλητές που θα συναντήσουμε είναι διακριτές. Βασικό χαρακτηριστικό του είναι ότι αυτά τα υπό-προβλήματα λύνονται κατά σειρά, ανεξάρτητα το ένα από τα υπόλοιπα. Σε αντίθεση με το γραμμικό προγραμματισμό δεν υπάρχει κάποια προκαθορισμένη, γενική διαδικασία επίλυσης που να καλύπτει όλο το φάσμα των προβλημάτων Δυναμικού Προγραμματισμού. Αυτό σημαίνει ότι πρέπει κάθε φορά να δημιουργούμε τις απαραίτητες σχέσεις και εξισώσεις που περιγράφουν το εκάστοτε πρόβλημα. Πρέπει δηλαδή να υπάρχει ένα καθορισμένο μοντέλο που να περιγράφει τη συμπεριφορά του προς μελέτη συστήματος. Ένα πλεονέκτημα των μεθόδων του Δυναμικού Προγραμματισμού είναι ότι χρησιμοποιούν λίγες υποθέσεις οι οποίες μπορούν μάλιστα να είναι μη-γραμμικές ή στοχαστικές. Αυτό έχει ως αποτέλεσμα να διευρύνεται το φάσμα των εφαρμογών του καθώς σε άλλες περιπτώσεις οι υποθέσεις είναι τόσο περιοριστικές, όπως για παράδειγμα η γραμμικότητα, που κάνουν τα προβλήματα είτε μη ρεαλιστικά είτε να απαιτούν άλλες μεθόδους επίλυσης.

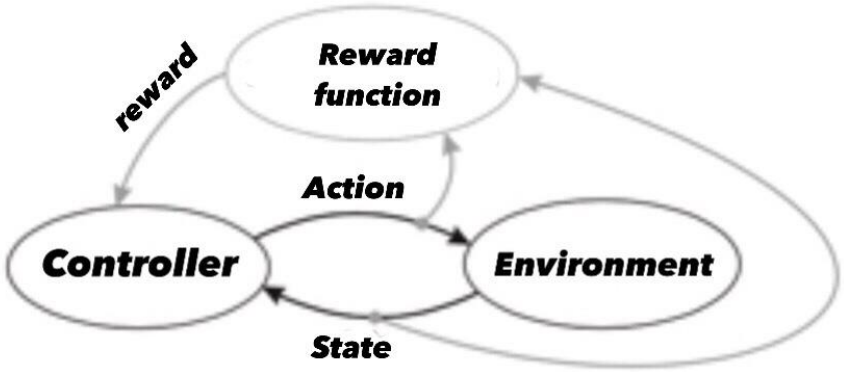
Θα περιγράψουμε με σύντομο τρόπο τη γενική διαδικασία με την οποία λειτουργεί ένα σύστημα Δυναμικού Προγραμματισμού. Υπάρχουν δύο βασικές βαθμίδες: [α] Ένας χειριστής (Controller) που έχει ως ρόλο τη λήψη των αποφάσεων του προβλήματος και [β] το Περιβάλλον (Environment) που αναπαριστά το πρόβλημα. Ο χειριστής που είναι υπεύθυνος για τις αποφάσεις προσπαθεί να επηρεάσει το περιβάλλον και το περιβάλλον με τη σειρά του αντιδρά σε αυτές τις αποφάσεις. Αυτή η αλληλεπίδραση γίνεται μέσω τριών παραγόντων:

1. Χώρος καταστάσεων (state space): αποτελείται από τις λεγόμενες «καταστάσεις» (states) και περιλαμβάνει όλες τις πιθανές μεταβλητές που μπορεί να λάβει το περιβάλλον.

2. Απόφαση (action): σε κάθε κατάσταση ο χειριστής θα πάρει μία απόφαση που έχει ως αποτέλεσμα το σύστημα να μεταβεί σε μία επόμενη κατάσταση μέσω της λεγόμενης «συνάρτησης μετάβασης» ή των «πιθανοτήτων μετάβασης». Ο χειριστής δηλαδή επηρεάζει το περιβάλλον μέσω αυτής της απόφασης και το περιβάλλον αλλάζει κατάσταση ως απάντηση στην απόφαση του.

3. Επιβράβευση ή κόστος (reward): το σύστημα δίνει στο χειριστή μία επιβράβευση, του παρέχει δηλαδή μία άμεση αξιολόγηση για το πόσο πολύ συνέβαλλε η τελευταία απόφαση στο να πραγματοποιηθεί από το περιβάλλον η διαδικασία που θέλουμε.

Στόχος της εκάστοτε τρέχουσας κατάστασης να επιλεγεί η βέλτιστη απόφαση η οποία μεγιστοποιεί τη μακροχρόνια επιβράβευση που παρέχεται από το περιβάλλον. Τελικός στόχος είναι η εύρεση μίας βέλτιστης πολιτικής.
Στο παρακάτω σχήμα παρουσιάζεται αυτή η κυκλική διαδικασία του Δυναμικού Προγραμματισμού.



Σχήμα 2.1: Κυκλική διαδικασία που ακολουθείται στο Δυναμικό Προγραμματισμό (Busoniu, Babuska, De Schutter, & Ernst, 2010)

Το πλαίσιο αναφοράς του Δυναμικού Προγραμματισμού βασίζεται στη διαδοχική επίλυση υπό-προβλημάτων όπως αναφέραμε παραπάνω. Αυτό σημαίνει ότι μία τέτοια διαδικασία όπως αυτή του σχήματος είναι η κεντρική ιδέα για κάθε ένα από αυτά τα υπό-προβλήματα. Στην επόμενη ενότητα παρουσιάζονται εκτενέστερα τα βασικά χαρακτηριστικά ενός προβλήματος Δυναμικού Προγραμματισμού και αναλύεται μία βασική μέθοδος επίλυσης τους.

2.2 ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΤΩΝ ΕΦΑΡΜΟΓΩΝ ΔΥΝΑΜΙΚΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Υπάρχουν κάποια βασικά χαρακτηριστικά που εμφανίζονται στα προβλήματα Δυναμικού προγραμματισμού{[1], [3], [5]}:

1. Διάσπαση του προβλήματος σε στάδια

Σε κάθε στάδιο χρειάζεται να πάρουμε μία απόφαση ώστε να βρούμε το επόμενο βέλτιστο στάδιο. Πολλές φορές είναι ευκολότερο τα στάδια να λυθούν οπισθοδρομικά, από το τέλος προς την αρχή, καθώς οι διαδρομές που οδηγούν από την παρούσα κατάσταση στο μελλοντικό βέλτιστο στόχο είναι πάντα ένα υποσύνολο όλων των διαδρομών που

ακολουθούν την κανονική προς τα εμπρός διαδρομή ξεκινώντας πάντα από την τρέχουσα κατάσταση. Αυτή η διαδικασία ονομάζεται «οπισθοδρομική αναδρομή», “backwards recursion” [5]. Συνεπώς, η τελική λύση του προβλήματος είναι άμεση συνέπεια των διαδοχικών επιλογών που κάναμε σε κάθε στάδιο.

2. Κάθε στάδιο αποτελείται από έναν αριθμό καταστάσεων (πεπερασμένο ή άπειρο)

Ουσιαστικά, οι καταστάσεις παρέχουν όλες τις απαραίτητες πληροφορίες προκειμένου να λυθεί το πρόβλημα του εκάστοτε σταδίου.

3. Η απόφαση σε κάθε στάδιο εξηγεί τον τρόπο με τον οποίο η κατάσταση στο παρόν στάδιο μετατρέπεται στην κατάσταση του επόμενου σταδίου

Οι μεταβάσεις από ένα στάδιο στο επόμενο μπορούν να είναι είτε ντετερμινιστικές (αιτιοκρατικές) ή στοχαστικές. Στην πρώτη περίπτωση, η κατάσταση του επόμενου σταδίου καθορίζεται πλήρως από την κατάσταση του σταδίου που βρισκόμαστε. Αντίθετα, στη περίπτωση της στοχαστικής διαδικασίας μετάβασης, η επόμενη κατάσταση καθορίζεται από κάποια τυχαία μεταβλητή. Όταν λοιπόν έχουμε μία τέτοια περίπτωση, δίνεται μία κατανομή πιθανότητας για το ποια θα είναι η κατάσταση του επόμενου σταδίου. Ωστόσο και αυτή η κατανομή πιθανότητας εξακολουθεί να καθορίζεται πλήρως από την κατάσταση και την απόφαση του τρέχοντος σταδίου. Ο χώρος καταστάσεων μαζί με τις πιθανές αποφάσεις και τις συναρτήσεις f_{ij} που θα ονομάσουμε στην πορεία «συνεισφορά απόφασης» αποτελούν αυτό που ονομάζουμε Μαρκοβιανή Διαδικασία Απόφασης (Markov Decision Process-MDP) [45].

4. Για δεδομένο τρέχον στάδιο, *η βέλτιστη πολιτική για τα υπολειπόμενα στάδια είναι ανεξάρτητη από τις πολιτικές που έχουν αποφασιστεί στα προηγούμενα στάδια.* Αυτό ονομάζεται «Αρχή της Βελτιστότητας στο Δυναμικό Προγραμματισμό» [1].

Με άλλα λόγια, οι μελλοντικές μας αποφάσεις επηρεάζονται μόνο από το παρόν (την τρέχουσα κατάσταση) και όχι από το παρελθόν, από το πώς βρεθήκαμε δηλαδή στην τρέχουσα κατάσταση. Αυτό μάλιστα αποτελεί και τον ορισμό ώστε να καλείται μια στοχαστική διαδικασία Μαρκοβιανή. Οποιοδήποτε πρόβλημα δεν ικανοποιεί την Αρχή της Βελτιστότητας δεν μπορεί να λυθεί μέσω του Δυναμικού Προγραμματισμού.

5. Η διαδικασία επίλυσης ξεκινάει βρίσκοντας τη βέλτιστη πολιτική για το τελευταίο στάδιο

Πιο συγκεκριμένα, επιλέγουμε τη βέλτιστη απόφαση προκειμένου να μεταβούμε στην κατάσταση του τελευταίου σταδίου από οποιαδήποτε πιθανή κατάσταση του προτελευταίου σταδίου. Συνήθως, αυτό το πρώτο βήμα είναι και το πιο εύκολο, καθώς πρόκειται για βελτιστοποίηση (μεγιστοποίηση ή ελαχιστοποίηση) ενός προβλήματος μίας μόνο μεταβλητής. Για τέτοιου είδους διευκολύνσεις άλλωστε, έχουμε επιλέξει να λύνουμε πολλά προβλήματα οπισθοδρομικά.

6. Υπάρχει μια αναδρομική σχέση που δίνει τη βέλτιστη απόφαση σε ένα στάδιο m δεδομένης της βέλτιστης απόφασης στο στάδιο m+1.

Η ακριβής αναδρομική σχέση κατασκευάζεται αναλόγως το εκάστοτε πρόβλημα. Ωστόσο οι κοινές μεταβλητές που παρουσιάζονται είναι οι εξής:

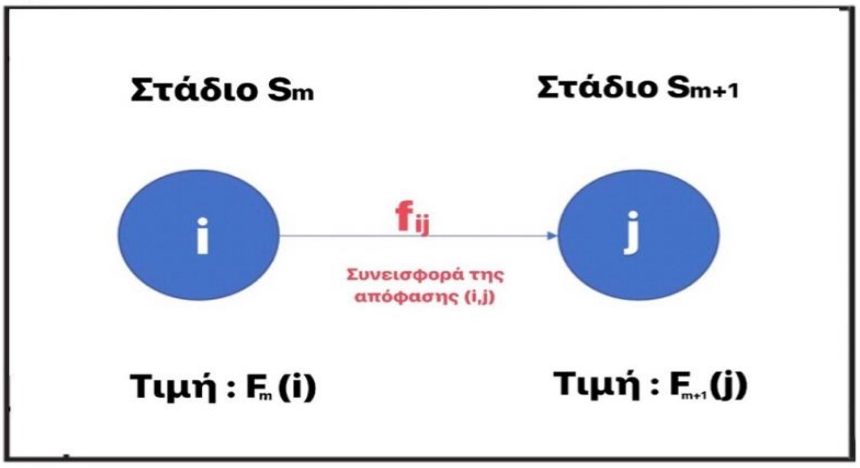
K : αριθμόςμγι σταδίων
s_m : τρέχον στάδιο (m=1, 2, …, K)
i : τρέχουσα κατάσταση σταδίου s_m
j : επόμενη κατάσταση σταδίου s_{m+1}
(i, j) : μεταβλητή απόφασης για το στάδιο m
f_{ij} : συνάρτηση επιβράβευσης (ή κόστους) για τη μετάβαση από την κατάσταση i ενός σταδίου m στην κατάσταση j του επόμενου σταδίου m+1
$F_m(i)$: συνάρτηση αθροιστικής επιβράβευσης (ή κόστους) για τη μετάβαση από την κατάσταση i ενός σταδίου m μέχρι την τελική κατάσταση ακολουθώντας τη βέλτιστη διαδρομή
Αντικειμενική συνάρτηση (objective function) : μία συνάρτηση, της οποίας αναζητείται το μέγιστο ή το ελάχιστο, δηλαδή περιγράφει το σκοπό κάθε προβλήματος βελτιστοποίησης

Χρησιμοποιώντας την αναδρομική σχέση ξεκινάμε από το τέλος βρίσκοντας τη βέλτιστη απόφαση για τη μετάβαση στην κατάσταση τέλους, όπως ακριβώς αναφέραμε και στο χαρακτηριστικό 5. Υπολογίζουμε δηλαδή τις συναρτήσεις $F_{K-1}(i)$ για κάθε κατάσταση i του σταδίου K-1 (προτελευταίο στάδιο). Στη συνέχεια προχωράμε με τον ίδιο τρόπο οπισθοδρομικά βρίσκοντας κάθε φορά τη βέλτιστη απόφαση (F_{K-2}, F_{K-3},…) για το εκάστοτε στάδιο K-2, K-3 κ.ο.κ. μέχρι να βρούμε τη βέλτιστη πολιτική του $1^{ου}$ σταδίου ($F_1(1)$). Αυτές οι βέλτιστες λύσεις που βρίσκουμε σε κάθε στάδιο συνθέτουν τη βέλτιστη πολιτική ολόκληρου του προβλήματος.

2.3. ΝΤΕΤΕΡΜΙΝΙΣΤΙΚΟΣ ΔΥΝΑΜΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

Στον Ντετερμινιστικό ή Αιτιοκρατικό Δυναμικό Προγραμματισμό (Deterministic Dynamic Programming) {[1], [3]} η κατάσταση του επόμενου σταδίου καθορίζεται πλήρως τόσο από την κατάσταση όσο και από την απόφαση που θα επιλέξουμε στο τρέχον στάδιο.

Στο σχήμα 2.2 παρουσιάζεται η βασική δομή των προβλημάτων Αιτιοκρατικού Δυναμικού Προγραμματισμού. Έστω ότι η διαδικασία βρίσκεται στην κατάσταση i κάποιου σταδίου s_m (m=1, 2, …, K). Παίρνοντας μία απόφαση (i, j) στο τρέχον στάδιο η διαδικασία μετακινείται στην κατάσταση j του επόμενου σταδίου s_{m+1}. Για να βρούμε, λοιπόν, την τιμή της αντικειμενικής συνάρτησης της κατάστασης i είναι απαραίτητο να συνυπολογίσουμε στη βέλτιστη τιμή $F_{m+1}(j)$ τη συνεισφορά της απόφασης (i, j), δηλαδή την f_{ij}. Αυτό οφείλεται στο γεγονός ότι λύνουμε το πρόβλημα οπισθοδρομικά. Το μόνο που μένει προκειμένου να βρούμε τη βέλτιστη απόφαση στην κατάσταση i που βρισκόμαστε, είναι η βελτιστοποίηση (μεγιστοποίηση ή ελαχιστοποίηση) ως προς τις δυνατές αποφάσεις που μπορούν να ληφθούν. Τέλος, αφού έχουμε υπολογίσει τη βέλτιστη ή τις βέλτιστες αποφάσεις και τις αντίστοιχες τιμές που δίνουν στην αντικειμενική συνάρτηση για όλες τις καταστάσεις του σταδίου s_m μπορούμε να προχωρήσουμε στην επίλυση του αμέσως προηγούμενου σταδίου s_{m-1} όπου θα ακολουθήσουμε την ίδια μεθοδολογία μέχρι να φτάσουμε στο αρχικό στάδιο.



Σχήμα 2.2 : Βασική δομή προβλημάτων Αιτιοκρατικού Δυναμικού Προγραμματισμού (Κολέτσος & Στογιάννης, 2021)

2.3.1. ΚΑΤΗΓΟΡΙΕΣ ΝΤΕΤΕΡΜΙΝΙΣΤΙΚΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Υπάρχουν 3 κατηγορίες προβλημάτων Ντετερμινιστικού Δυναμικού Προγραμματισμού οι οποίες διακρίνονται [3] :

I. Βάσει του τύπου της αντικειμενικής συνάρτησης
   a. Ελαχιστοποίηση του αθροίσματος των συνεισφορών απόφασης (f_{ij}) σε κάθε στάδιο
   b. Μεγιστοποίηση του αθροίσματος των συνεισφορών απόφασης (f_{ij}) σε κάθε στάδιο
   c. Ελαχιστοποίηση του γινομένου των συνεισφορών απόφασης (f_{ij}) σε κάθε στάδιο
   d. Μεγιστοποίηση του γινομένου των συνεισφορών απόφασης (f_{ij}) σε κάθε στάδιο

II. Βάσει της μορφής των καταστάσεων
   a. Αν περιγράφονται από διακριτές μεταβλητές
   b. Αν περιγράφονται από συνεχείς μεταβλητές

III. Βάσει των μεταβλητών απόφασης
   a. Αν είναι διακριτές μεταβλητές
   b. Αν είναι συνεχείς μεταβλητές

2.4. ΚΛΑΣΕΙΣ ΠΟΛΥΠΛΟΚΟΤΗΤΑΣ

Τα προβλήματα με τα οποία ασχολείται η Ε.Ε. λύνονται με τη βοήθεια αλγορίθμων οι οποίοι χωρίζονται σε κατηγορίες αναλόγως την αποδοτικότητά τους. Ένα μέτρο αποδοτικότητας αποτελεί η θεωρία πολυπλοκότητας μέσω της οποίας εκτιμώνται τόσο ο χρόνος (run time) όσο και ο χώρος (space) δηλαδή η μνήμη που χρειάζεται ένας βέλτιστος αλγόριθμος προκειμένου να λύσει ένα πρόβλημα. Αυτές οι δύο έννοιες ονομάζονται time και space complexity, αντίστοιχα. Τα προβλήματα χωρίζονται με τη σειρά τους σε κλάσεις πολυπλοκότητας όπου σε κάθε κλάση το σετ των προβλημάτων έχουν την ίδια δυσκολία επίλυσης [2].

1. Κλάση προβλημάτων P

Τα προβλήματα της κλάσης P [2] μπορούν να λυθούν σε πολυωνυμικό χρόνο (polynomial time) από έναν ντετερμινιστικό αλγόριθμο (ή ντετερμινιστική μηχανή Turing[2]). Παράδειγμα αποτελεί το πρόβλημα ελάχιστης διαδρομής (κεφάλαιο 3).

2. Κλάση προβλημάτων NP (Nondeterministic Polynomial)

Η κλάση {[2], [8]} αυτή χαρακτηρίζεται από προβλήματα απόφασης[3] των οποίων οι λύσεις είναι δύσκολο να βρεθούν αλλά εύκολο να επιβεβαιωθούν. Με άλλα λόγια λύνονται σε πολυωνυμικό χρόνο από μη ντετερμινιστικούς αλγορίθμους (ή μη ντετερμινιστική μηχανή Turing) και οι λύσεις της μορφής «ναι» ή «όχι» μπορούν να επαληθευτούν σε πολυωνυμικό χρόνο από έναν ντετερμινιστικό αλγόριθμο. Η κλάση αυτή, είναι υπερσύνολο της κλάσης P καθώς αν ένα πρόβλημα μπορεί να επιλυθεί σε πολυωνυμικό χρόνο (κλάση P) τότε και η λύση του μπορεί να επιβεβαιωθεί σε πολυωνυμικό χρόνο (κλάση NP). Παρολ'αυτά η κλάση NP περιλαμβάνει και πολλά ακόμα προβλήματα όπως αυτά της παρακάτω κλάσης.

3. Κλάση προβλημάτων NP-Complete

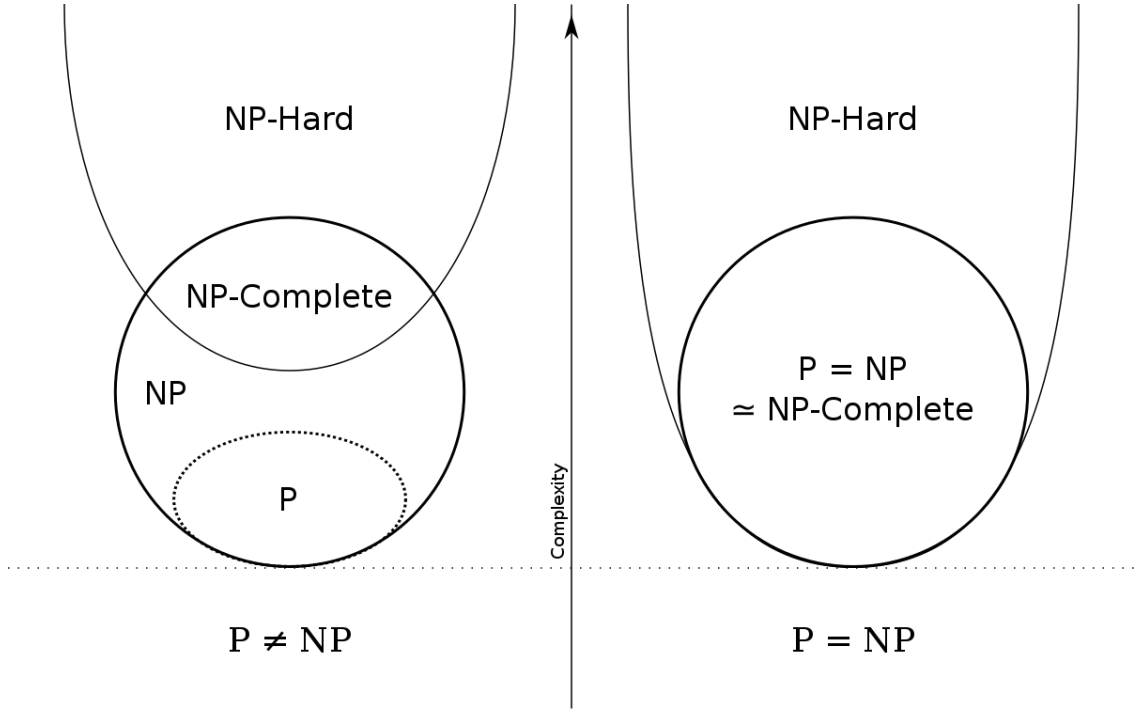
Τα πιο δύσκολα προβλήματα απόφασης της κλάσης NP ανήκουν στην κλάση NP-Complete {[2], [6], [8]}. Για αυτά δεν έχει αποδειχθεί ακόμα ότι είναι επιλύσιμα σε πολυωνυμικό χρόνο αλλά ούτε έχει αποδειχθεί ότι δεν υπάρχει κάποιος αλγόριθμος που να τα επιλύει σε πολυωνυμικό χρόνο. Το πρόβλημα αυτό, ότι δεν είναι δηλαδή γνωστό αν αυτά τα προβλήματα μπορούν να λυθούν γρήγορα ονομάζεται «*P versus NP Problem»* [8] και είναι ένα από τα πιο θεμελιώδη άλυτα προβλήματα στην επιστήμη των υπολογιστών σήμερα. Αν αποδειχθεί ότι κάποιο πρόβλημα NP-Complete μπορεί να λυθεί σε πολυωνυμικό χρόνο θα έχει αποδειχθεί ότι μπορούν όλα. Θα έχει αποδειχθεί δηλαδή ότι P=NP. Μέχρι στιγμής πιστεύεται ότι P $\neq NP$.
Για να αποδειχθεί ότι ένα πρόβλημα X είναι NP-Complete αρκεί να πάρουμε ένα γνωστό NP-Complete πρόβλημα και να το μετασχηματίσουμε, έτσι ώστε ο αλγόριθμος του ήδη γνωστού προβλήματος να μπορεί να γίνει κομμάτι του αλγορίθμου του X. Το πρώτο πρόβλημα που αποδείχθηκε ότι είναι NP-Complete είναι το SAT (Boolean satisfiability problem) από τον Cook [6].

[2] Μαθηματικό υπολογιστικό μοντέλο με πεπερασμένο αριθμό καταστάσεων, το οποίο μπορεί να προσομοιώσει οποιοδήποτε αλγόριθμο [7].
[3] Προβλήματα τα οποία δέχονται απάντηση της μορφής ναι ή όχι [2].

4. Κλάση προβλημάτων NP-Hard

Ένα πρόβλημα Χ, όχι απαραίτητα απόφασης, είναι NP-Hard {[2], [8]} αν κάθε πρόβλημα κλάσης NP μπορεί να αναχθεί στο Χ σε πολυωνυμικό χρόνο. Κατά συνέπεια, αν ίσχυε ότι P $\neq NP$, αυτό θα σήμαινε ότι τα NP-Hard προβλήματα δεν μπορούν να επιλυθούν σε πολυωνυμικό χρόνο. Ένα τέτοιο παράδειγμα αποτελεί το πρόβλημα του πλανόδιου πωλητή (κεφάλαιο 4).

Οι κλάσεις συνοψίζονται στο παρακάτω σχήμα:



Σχήμα 2.4.1: Διάγραμμα Euler για κλάσεις προβλημάτων P, NP, NP-Complete, NP-Hard υπό τις εκάστοτε υποθέσεις P $\neq NP$ και P=NP, αντίστοιχα [9].

ΚΕΦΑΛΑΙΟ 3

3.1. ΑΠΟΣΤΑΣΗ ΚΑΤΑ LEVENSHTEIN (LEVENSHTEIN DISTANCE)

Το πρόβλημα απόστασης κατά Levenshtein χαρακτηρίζεται ως «το πιο κλασικό, πρόβλημα ασαφούς αντιστοίχισης του Δυναμικού προγραμματισμού» [10]. Είναι γνωστό με πολλά διαφορετικά ονόματα όπως “Edit Distance Problem” ή “String Editing Problem”. Στόχος είναι η μετατροπή μίας συμβολοσειράς σε μία άλλη συμβολοσειρά μέσω 3 δυνατών διαδικασιών (διαγραφής, προσθήκης, αντικατάστασης) με το μικρότερο δυνατό κόστος. Το πρόβλημα αυτό έχει εφαρμογή στην επεξεργασία φυσικής γλώσσας, για τον έλεγχο ορθογραφίας λέξεων και κατ’ επέκταση για ρυθμίσεις αυτόματης διόρθωσης, όπου για τη διόρθωση μίας λέξης επιλέγονται από ένα μεγάλο λεξικό υποψήφιες λέξεις που έχουν μικρό κόστος μετατροπής, δηλαδή εκείνες που είναι πολύ κοντά στην αρχική λέξη που ελέγχεται. Για παράδειγμα η λέξη «και» μετατρέπεται στη λέξη «ναι» μέσω μίας αντικατάστασης του γράμματος «κ» με το γράμμα «ν». Η λέξη «γεια» μετατρέπεται σε «για» μέσω αφαίρεσης του «ε» κ.ο.κ. Επίσης, στην επιστήμη της βιολογίας που συνδυάζεται με τον κλάδο της πληροφορικής για την επεξεργασία μεγάλων όγκων δεδομένων (bioinformatics), το πρόβλημα αυτό βρίσκει εφαρμογή στη σύγκριση αλυσίδων DNA και μπορεί να συμβάλει σε πληθώρα εφαρμογών από την ανίχνευση γενετικών ανωμαλιών μέχρι τη σύγκριση λειτουργιών και χαρακτηριστικών διαφορετικών ειδών που παρουσιάζουν ομοιότητες στο DNA τους.

3.2. ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ

Θεωρούμε Σ ένα σύνολο χαρακτήρων, δηλαδή μία αλφάβητο. Δίνονται δύο συμβολοσειρές x, y έστω x =x[1…m] και y =y[1…n] όπου $x \in \Sigma^m$ και $y \in \Sigma^n$. Στόχος του προβλήματος είναι η μετατροπή της συμβολοσειράς x στη συμβολοσειρά y χρησιμοποιώντας τον μικρότερο αριθμό εκ των τριών δυνατών διαδικασιών [10]:

1. **Προσθήκη (I)** χαρακτήρα από το σύνολο Σ στη συμβολοσειρά x με κόστος c(I).
2. **Αφαίρεση (D)** χαρακτήρα από τη συμβολοσειρά x με κόστος c(D).
3. **Αντικατάσταση (R)** ενός χαρακτήρα της συμβολοσειράς x με ένα χαρακτήρα από το σύνολο Σ.

Αυτές οι διαδικασίες εφαρμόζονται στη συμβολοσειρά x, δηλαδή σε αυτή που θέλουμε να μετατρέψουμε. Το συνολικό πρόβλημα της μετατροπής αυτής μπορεί να διασπαστεί σε υπό-προβλήματα, ώστε να γίνονται συνεχείς μετατροπές της συμβολοσειράς μικρότερου μήκους x[1…i] στη συμβολοσειρά y[1…j] έως ότου ολοκληρωθεί η μετατροπή. Συμβολίζουμε f(i, j) το ελάχιστο κόστος που προκύπτει από την εφαρμογή αυτών των διαδικασιών. Χωρίζουμε το πρόβλημα σε τρεις βασικές περιπτώσεις [11].

1η περίπτωση: *Χαρακτήρας του x δεν αντιστοιχεί σε χαρακτήρα του y*

Όταν μελετάμε συμβολοσειρές διαφορετικού μήκους μπορεί η μια συμβολοσειρά να υπολείπεται σε χαρακτήρες από την άλλη. Σε αυτή την περίπτωση ο χαρακτήρας μίας συμβολοσειράς που δεν αντιστοιχεί σε χαρακτήρα της άλλης συμβολοσειράς θεωρείται κενός. Διακρίνουμε δύο περιπτώσεις:

Α. Αν η συμβολοσειρά x είναι κενή τότε προσθέτουμε όλους τους εναπομείναντες χαρακτήρες του y στο x και το κόστος αυτής της διαδικασίας ισούται με το πλήθος των χαρακτήρων του y που προτέθηκαν. Δηλαδή, f(0, j)= j.

(“ “, ‘’$y_1y_2y_3$’’) → (’’ $y_1y_2y_3$’’, ‘’ $y_1y_2y_3$’’) με κόστος = 3

Β. Αν η συμβολοσειρά y είναι κενή τότε αφαιρούμε όλους τους εναπομείναντες χαρακτήρες από τη συμβολοσειρά x και το κόστος της διαδικασίας ισούται με το πλήθος των χαρακτήρων του x που αφαιρέθηκαν. Δηλαδή, f(i, 0) = i

(“$x_1x_2x_3$”, “ ”) → (“ ”, “ ”) με κόστος = 3

2η περίπτωση: *Ίδιοι τελευταίοι χαρακτήρες*

Στην περίπτωση αυτή δεν εφαρμόζεται κανένας από τους τρεις τύπους αλλαγών και συνεχίζουμε τη διαδικασία αγνοώντας τον τελευταίο χαρακτήρα. Δηλαδή,

(“$x_1x_2x_3$”, “$y_1y_2y_3$”) με $x_3 = y_3$ → (“x_1x_2”, “y_1y_2”) με κόστος 0.

3η περίπτωση: *Διαφορετικοί τελευταίοι χαρακτήρες*

Στην περίπτωση αυτή βρίσκουμε το ελάχιστο κόστος εφαρμόζοντας μία εκ των τριών διαδικασιών:

Α. Προσθήκη του τελευταίου χαρακτήρα του y στο x.

Τότε το μήκος της συμβολοσειράς x παραμένει ίδιο ενώ το μήκος της συμβολοσειράς y μειώνεται κατά 1. Έχουμε δηλαδή x[1…i] και y[1…j-1]. Πράγματι,

(“$x_1x_2x_3$”, “$y_1y_2y_3$”) με $x_3 \neq y_3$ → (“$x_1x_2x_3$**y_3**” , “y_1y_2**y_3**”) = (“$x_1x_2x_3$”, “y_1y_2”) (από 2η περίπτωση)

με κόστος f(i, j-1) + c(I).

Β. Αφαίρεση του τελευταίου χαρακτήρα του x

Τότε το μήκος της συμβολοσειράς x μειώνεται κατά 1 ενώ το μήκος της συμβολοσειράς y παραμένει ίδιο. Έχουμε δηλαδή x[1…i-1], y[1…j]. Πράγματι,

(“$x_1x_2x_3$”, “$y_1y_2y_3$”) με $x_3 \neq y_3$ → (“x_1x_2”, “$y_1y_2y_3$”) με κόστος f(i-1, j) + c(D).

Γ. Αντικατάσταση τρέχοντος χαρακτήρα του x με τον αντίστοιχο τρέχων χαρακτήρα του y.

Τότε το μήκος και των δύο συμβολοσειρών x, y μειώνεται κατά 1. Έχουμε δηλαδή x[1…i-1], y[1…j-1]. Πράγματι,

(“$x_1x_2x_3$”, “$y_1y_2y_3$”) με $x_3 \neq y_3$ → (“x_1x_2**y_3**”, “y_1y_2**y_3**”)= (“x_1x_2”, “y_1y_2”) (από 2η περίπτωση)

με κόστος f(i-1, j-1) + c(R) όπου c(R) = $\begin{cases} 0, & \alpha\nu\ xi = yj \\ c(R), & \alpha\nu\ xi \neq yj \end{cases}$

Δηλαδή αν οι χαρακτήρες είναι οι ίδιοι τότε δεν πραγματοποιείται αντικατάσταση, ενώ αν είναι διαφορετικοί τότε απαιτείται αντικατάσταση.

Συνολικά λοιπόν η αναδρομική σχέση [10] που περιγράφει το πρόβλημα είναι:

$$f(i,j) = \begin{cases} j, & \alpha\nu\ i = 0 \\ i, & \alpha\nu\ j = 0 \\ \min\{ f(i,j-1) + c(I), f(i-1,j) + c(D) + f(i-1,j-1) + c(R), & \alpha\nu\ i,j > 0 \end{cases} \quad (3.1)$$

Όπου τα κόστη είναι τα εξής:

$$\text{C} = \begin{cases} c(I), & \alpha\nu\ \pi\rho o\sigma\tau\epsilon\theta\epsilon\acute{\iota}\ \acute{\epsilon}\nu\alpha\varsigma\ \chi\alpha\rho\alpha\kappa\tau\acute{\eta}\rho\alpha\varsigma\ \sigma\epsilon\ o\pi o\iota\alpha\delta\acute{\eta}\pi o\tau\epsilon\ \theta\acute{\epsilon}\sigma\eta\ \tau o\upsilon\ x \\ c(D), & \alpha\nu\ \alpha\phi\alpha\iota\rho\epsilon\theta\epsilon\acute{\iota}\ \acute{\epsilon}\nu\alpha\varsigma\ \chi\alpha\rho\alpha\kappa\tau\acute{\eta}\rho\alpha\varsigma\ \ \alpha\pi\acute{o}\ o\pi o\iota\alpha\delta\acute{\eta}\pi o\tau\epsilon\ \theta\acute{\epsilon}\sigma\eta\ \tau o\upsilon\ x \\ c(R) = \begin{cases} 0, & \alpha\nu\ xi = yj\ \ \delta\epsilon\nu\ \pi\rho\alpha\gamma\mu\alpha\tau o\pi o\iota\epsilon\acute{\iota}\tau\alpha\iota\ \alpha\nu\tau\iota\kappa\alpha\tau\acute{\alpha}\sigma\tau\alpha\sigma\eta \\ c(R), & \alpha\nu\ xi \neq yj\ \ \pi\rho\alpha\gamma\mu\alpha\tau o\pi o\iota\epsilon\acute{\iota}\tau\alpha\iota\ \alpha\nu\tau\iota\kappa\alpha\tau\acute{\alpha}\sigma\tau\alpha\sigma\eta \end{cases} \end{cases} \quad (3.2)$$

Στη γενικότερη περίπτωση καθεμία από τις διαδικασίες αυτές μπορεί να έχει διαφορετικό βάρος ωστόσο στην απλή περίπτωση με την οποία θα ασχοληθούμε θεωρούμε τις αλλαγές ισότιμες. Ακόμη, χωρίς βλάβη της γενικότητας θεωρούμε τα κόστη ίσα με τη μονάδα οπότε η αναδρομική σχέση παίρνει την τελική μορφή:

$$f(i,j) = \begin{cases} j, & \alpha\nu\ i = 0 \\ i, & \alpha\nu\ j = 0 \\ \min\{ f(i,j-1) + 1, f(i-1,j) + 1, f(i-1,j-1) + c(R), & \alpha\nu\ i,j > 0 \end{cases} \quad (3.3)$$

Όπου c = $\begin{cases} 1 \\ 1 \\ \begin{cases} 0, \alpha\nu\ xi = yj \\ 1, \alpha\nu\ xi \neq yj \end{cases} \end{cases}$ (3.4)

Συνεπώς, η επίλυση ξεκινάει από μικρότερα υπό-προβλήματα καταλήγοντας στο συνολικό πρόβλημα, δηλαδή την πλήρη μετατροπή της συμβολοσειράς x στη συμβολοσειρά y. Στο πλαίσιο αυτό, μπορούμε να δημιουργήσουμε έναν πίνακα ο οποίος εφαρμόζοντας την αναδρομική σχέση (3.3) αποθηκεύει όλους τους υπολογισμούς και καταλήγει στο τελικό ελάχιστο κόστος, δηλαδή στη συνολική απόσταση Levenshtein.

Έστω οι συμβολοσειρές x, y μήκους m, n, αντίστοιχα. Δημιουργούμε έναν πίνακα και ακολουθούμε την παρακάτω διαδικασία {[12], [13]}:

	" "	X1	X2	X3	…	Xm
" "						
Y1		αντικατάσταση	προσθήκη			
Y2		αφαίρεση	τρέχουσα κατάσταση f[i][j]			
Y3						
…						
Yn						τελικό ελαχιστο κόστος

Πίνακας 3.2.1: Απεικόνιση προβλήματος απόστασης Levenshtein μέσω κατασκευής πίνακα

Ο πίνακας γεμίζει γραμμή προς γραμμή. Κάθε κελί συμβολίζει μία κατάσταση. Σε κάθε βήμα, τα μόνα αποτελέσματα που χρειάζονται είναι αυτά που φαίνονται στο κουτί του πίνακα με το έντονο περίγραμμα. Δηλαδή, η αριστερή, η πάνω και η διαγώνια θέση. Καθεμία, από αυτές συμβολίζει την αφαίρεση, την προσθήκη και την αντικατάσταση ενός χαρακτήρα, αντίστοιχα. Η τιμή της τρέχουσας κατάστασης εξαρτάται μόνο από αυτά τα τρία γειτονικά κελιά, η τιμή των οποίων έχει ήδη υπολογιστεί σε προηγούμενο βήμα. Το f[i][j] παίρνει την τιμή του γειτονικού κελιού που αντιστοιχεί στο ελάχιστο κόστος συν μία[4] μονάδα για την αλλαγή που θα πραγματοποιηθεί (3.4). Αν i=j τότε το f[i][j] παίρνει αυτόματα την τιμή του διαγώνιου κελιού καθώς οι χαρακτήρες ταυτίζονται και δεν απαιτείται καμία επιπλέον μετατροπή, δηλαδή το επιπλέον κόστος είναι 0. Στο κάτω δεξί κελί του πίνακα φαίνεται η τελική λύση που αντιστοιχεί στη συνολική απόσταση Levenshtein, δηλαδή στο ελάχιστο κόστος που απαιτείται για τη μετατροπή του x σε y. Με οπισθοδρομικό τρόπο γυρνάμε στην αρχή ώστε να καταλήξουμε στον τρόπο με τον οποίο φτάσαμε στη βέλτιστη μετατροπή. Αυτή η μέθοδος θα παρουσιαστεί στην επόμενη ενότητα παραθέτοντας ένα παράδειγμα.

3.2.1. ΠΟΛΥΠΛΟΚΟΤΗΤΑ ΠΡΟΒΛΗΜΑΤΟΣ EDIT DISTANCE

Το πρόβλημα αυτό θεωρητικά[5] ανήκει στην κατηγορία κλάσης P δηλαδή η επίλυση του γίνεται σε πολυωνυμικό χρόνο μέσω ντετερμινιστικού αλγορίθμου [14]. Επιλέγοντας ως τρόπο επίλυσης μεθόδους δυναμικού προγραμματισμού μειώνεται κατά πολύ ο χρόνος και ο χώρος πολυπλοκότητας. Πιο συγκεκριμένα, λύνονται $m \cdot n$ υποπροβλήματα όπου m, n το μήκος της συμβολοσειράς x και y, αντίστοιχα. Χρειάζεται δηλαδή $O(m \cdot n)$ χρόνος. Ακόμη, θα πρέπει να γίνει αποθήκευση των $m \cdot n$ υποπροβλημάτων επομένως χρειάζεται $O(m \cdot n)$ χώρος. Ωστόσο εφόσον προσθέτουμε μία επιπλέον γραμμή και στήλη ο χώρος είναι $O(m \cdot n)$ Τέλος για την οπισθοδρόμηση που θα οδηγήσει στη βέλτιστη λύση χρειάζεται $O(m + n)$ χρόνος [12].

[4] Αν είχαμε θέσει διαφορετικά βάρη για κάθε διαδικασία (I, D, R) τότε θα προσθέταμε το εκάστοτε βάρος.

[5] Πρακτικά, σε ρεαλιστικά παραδείγματα όπως η σύγκριση δισεκατομμυρίων γονιδιωμάτων σε αλυσίδες DNA μπορεί να πάρει χιλιάδες χρόνια ώστε να τρέξει ένας τέτοιος αλγόριθμος.

3.3. ΠΑΡΑΔΕΙΓΜΑ

Δεν είναι λίγες οι φορές κατά τις οποίες γράφοντας ένα κείμενο η αυτόματη διόρθωση αντικαθιστά λανθασμένα τη σωστή λέξη που θέλαμε να χρησιμοποιήσουμε με κάποια άλλη που παρουσιάζει αρκετές ομοιότητες είτε συνολικά, είτε επειδή ξεκινάει από τα ίδια γράμματα. Θα παρουσιάσουμε ένα παράδειγμα δύο λέξεων με κοινή αρχή προκειμένου να εντοπίσουμε τον ελάχιστο αριθμό αλλαγών ώστε να μετατρέψουμε τη μία λέξη στην άλλη.

Θέλουμε να μετατρέψουμε τη λέξη **«ευχαριστώ»** στη λέξη **«ευκαιρία»**. Δημιουργούμε ανάλογο πίνακα με τον πίνακα 3.2.1 και υπολογίζοντας την πρώτη γραμμή και στήλη έχουμε:

		X									
		0	1	2	3	4	5	6	7	8	9
		" "	Ε	Υ	Χ	Α	Ρ	Ι	Σ	Τ	Ω
	0 " "	0	1	2	3	4	5	6	7	8	9
	1 Ε	1									
	2 Υ	2									
	3 Κ	3									
Υ	4 Α	4									
	5 Ι	5									
	6 Ρ	6									
	7 Ι	7									
	8 Α	8									

Πίνακας 3.3.1: Υπολογισμοί πρώτης γραμμής και πρώτης στήλης

Για την συμπλήρωση της πρώτης γραμμής και στήλης βρισκόμαστε στην 1η περίπτωση που αναλύσαμε παραπάνω. Δηλαδή παρουσιάζεται κενό και στις 2 λέξεις.

Όταν βρισκόμαστε στην αρχική κατάσταση, κελί [0,0], δεν απαιτείται καμία αλλαγή οπότε έχουμε κόστος 0.
Όταν βρισκόμαστε στο κελί [1,0] η μετατροπή που θέλουμε να πραγματοποιηθεί είναι η εξής:

“Ε” → “ “ : δηλαδή να μετατρέψουμε το γράμμα “Ε” στο κενό. Αυτό γίνεται μέσω μίας αφαίρεσης, του “Ε” και έτσι από αναδρομική σχέση (3.3) έχουμε f(1,0) = 1

Όταν βρισκόμαστε στο κελί [2,0] η μετατροπή που θέλουμε να συμβεί είναι η εξής:

“ΕΥ” → “ “ : δηλαδή να μετατρέψουμε το ΕΥ στο κενό. Αυτό γίνεται μέσω δύο αφαιρέσεων, του “Ε” και του “Υ” και έτσι από αναδρομική σχέση (3.3) έχουμε f(2,0) = 2.

Όμοια, συμπληρώνεται η υπόλοιπη γραμμή όπου εφαρμόζεται η σχέση f(i,0)=i

Αντίστοιχα, όταν βρισκόμαστε στο κελί [0,1] η μετατροπή που θέλουμε να συμβεί είναι η εξής:

“ “ → “Ε” : δηλαδή να μετατρέψουμε το κενό στο γράμμα “Ε”. Αυτό γίνεται μέσω μίας προσθήκης και από αναδρομική σχέση έχουμε f(0,1) = 1

Όταν βρισκόμαστε στο κελί [0,2] η μετατροπή που θέλουμε να πραγματοποιηθεί είναι η εξής:
“ΕΥ” → “ “ : δηλαδή να μετατρέψουμε το κενό σε “ΕΥ”. Αυτό γίνεται μέσω δύο προσθηκών και από αναδρομική σχέση έχουμε f(0,2) = 2

Όμοια, συμπληρώνεται η υπόλοιπη στήλη όπου εφαρμόζεται η σχέση f(0, j)= j.

Στο επόμενο βήμα συμπληρώνεται η επόμενη γραμμή ως εξής:

			X								
			0	1	2	3	4	5	6	7	8	9
			" "	E	Y	X	A	P	I	Σ	T	Ω
	0	" "	0	1	2	3	4	5	6	7	8	9
	1	E	1	0	1	2	3	4	5	6	7	8
	2	Y	2									
	3	K	3									
Y	4	A	4									
	5	I	5									
	6	P	6									
	7	I	7									
	8	A	8									

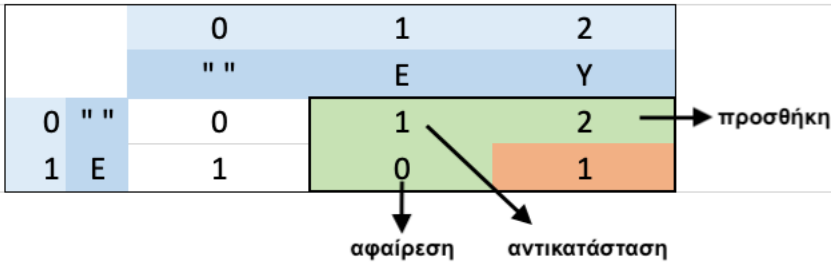
Πίνακας 3.3.2: Υπολογισμοί δεύτερης γραμμής. Τα δύο χρωματισμένα κελιά αντιστοιχούν σε εκείνα των οποίων οι υπολογισμοί τους αναλύονται.

Στο κελί [1,1] εξετάζουμε την εξής μετατροπή:

“E” → “E”. Εφόσον οι χαρακτήρες είναι οι ίδιοι δεν χρειάζεται καμία μετατροπή και όπως αναφέραμε στην παραπάνω ενότητα συμπληρώνουμε απευθείας με την τιμή του διαγώνιου κελιού.

Στο κελί [2,1] εξετάζουμε την εξής μετατροπή:

“EY” → “E”. Οι τελευταίοι χαρακτήρες δεν ταυτίζονται οπότε η τιμή του κελιού εξαρτάται από τα τρία γειτονικά κελία, αυτά με πράσινο χρώμα στον παρακάτω πίνακα.



Πίνακας 3.3.3: Εύρεση τιμής κελιού [2,1] (πορτοκαλί χρώμα) βρίσκοντας την ελάχιστη τιμή των γειτονικών κελιών (πράσινο χρώμα)

f[2][1] = min{ προσθήκη + 1, αφαίρεση + 1, αντικατάσταση + 1} = min{ 2+1, 0+1, 1+1} = 1

Με τον ίδιο τρόπο συμπληρώνονται όλες οι γραμμές του πίνακα και τελικά έχουμε:

			X									
			0	1	2	3	4	5	6	7	8	9
			" "	E	Y	X	A	P	I	Σ	T	Ω
	0	" "	0	1	2	3	4	5	6	7	8	9
	1	E	1	0	1	2	3	4	5	6	7	8
	2	Y	2	1	0	1	2	3	4	5	6	7
	3	K	3	2	1	1	2	3	4	5	6	7
Y	4	A	4	3	2	2	1	2	3	4	5	6
	5	I	5	4	3	3	2	2	2	3	4	5
	6	P	6	5	4	4	3	2	3	3	4	5
	7	I	7	6	5	5	4	3	2	3	4	5
	8	A	8	7	6	6	5	4	3	3	4	5

Πίνακας 3.3.4: Συνολικός πίνακας υπολογισμών. Με κίτρινο χρώμα φαίνεται το συνολικό ελάχιστο κόστος

Μέσω οπισθοδρόμησης βρίσκουμε τη διαδρομή που αντιστοιχεί στη βέλτιστη ακολουθία διαδικασιών ξεκινώντας από το τελικό κόστος φτάνοντας στην αρχή. Σε κάθε προς τα πίσω βήμα βρίσκουμε ποια ήταν η ελάχιστη τιμή που μας οδήγησε στην επόμενη κατάσταση του επόμενου βήματος. Η προς τα πίσω διαδρομή φαίνεται στον παρακάτω πίνακα:

			X									
			0	1	2	3	4	5	6	7	8	9
			" "	E	Y	X	A	P	I	Σ	T	Ω
	0	" "	0	1	2	3	4	5	6	7	8	9
	1	E	1	0	1	2	3	4	5	6	7	8
	2	Y	2	1	0	1	2	3	4	5	6	7
	3	K	3	2	1	1	2	3	4	5	6	7
Y	4	A	4	3	2	2	1	2	3	4	5	6
	5	I	5	4	3	3	2	2	2	3	4	5
	6	P	6	5	4	4	3	2	3	3	4	5
	7	I	7	6	5	5	4	3	2	3	4	5
	8	A	8	7	6	6	5	4	3	3	4	5

Πίνακας 3.3.5: Απεικόνιση οπισθοδρομικής διαδρομής που αντιστοιχεί στο βέλτιστο edit distance

Για να φτάσουμε στην κατάσταση [9,8], η βέλτιστη απόφαση είναι η αντικατάσταση χαρακτήρα. Για να φτάσουμε στην κατάσταση [8,7], η βέλτιστη απόφαση είναι πάλι η αντικατάσταση χαρακτήρα. Με τον ίδιο τρόπο συνεχίζουμε μέχρι την αρχική κατάσταση [0,0].

Από το τέλος προς την αρχή η ακολουθία διαδικασιών είναι:

RRRMDMRMM όπου R= αντικατάσταση (replace), M=ταύτιση (match), D=διαγραφή (delete)

Αντιστρέφεται η διαδρομή ώστε να ξεκινάει από την αρχική κατάσταση [0,0] και να φτάνει μέχρι την τελική κατάσταση [9,8]. Τελικά, η πλήρης μετατροπή της λέξης «ευχαριστώ» στη λέξη «ευκαιρία» απεικονίζεται με πράσινο χρώμα στον παρακάτω πίνακα.

			X								
		0	1	2	3	4	5	6	7	8	9
		" "	E	Y	X	A	P	I	Σ	T	Ω
	0	" "	0	1	2	3	4	5	6	7	8	9
	1	E	1	0	1	2	3	4	5	6	7	8
	2	Y	2	1	0	1	2	3	4	5	6	7
	3	K	3	2	1	1	2	3	4	5	6	7
Y	4	A	4	3	2	2	1	2	3	4	5	6
	5	I	5	4	3	3	2	2	2	3	4	5
	6	P	6	5	4	4	3	2	3	3	4	5
	7	I	7	6	5	5	4	3	2	3	4	5
	8	A	8	7	6	6	5	4	3	3	4	5

Πίνακας 3.3.6: Η βέλτιστη πορεία των διαδικασιών για τη μετατροπή της λέξης «ευχαριστώ» στη λέξη «ευκαιρία» απεικονίζεται με πράσινο χρώμα

Η μετατροπή είναι:

E	Y	X	A	P	I	Σ	T	Ω
|	|	|	|	|	|	|	|	|
E	Y	K	A	-	I	P	I	A

Μέσω των εξής διαδικασιών κατά σειρά:

MMRMDMRRR

Χρόνος πολυπλοκότητας: $O(9 \cdot 8)$
Χώρος πολυπλοκότητας: $O(9 \cdot 8)$

3.4. ΑΠΟΣΤΑΣΗ ΚΑΤΑ LEVENSHTEIN ΚΑΙ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

Όπως αναφέρθηκε στην προηγούμενη ενότητα ένας από τους τομείς στους οποίους εφαρμόζεται το πρόβλημα της απόστασης κατά Levenshtein είναι η επεξεργασία φυσικής γλώσσας για ασαφείς αντιστοιχίσεις (fuzzy matching). Τα συστήματα που χρησιμοποιούν κατά κύριο λόγο τον αλγόριθμο του edit distance έχουν να κάνουν με ορθογραφικούς ελέγχους, αυτόματες διορθώσεις κειμένου και μηχανές αναζήτησης όπου γίνονται προτάσεις που πλησιάζουν αυτό που έγραψε ο χρήστης. Οι προτάσεις δηλαδή που θα εμφανιστούν τελικά έχουν προέλθει από εφαρμογή του αλγορίθμου και επιλέγονται αυτές με το μικρότερο edit distance. Χαρακτηριστικό παράδειγμα αποτελεί η συνήθης χρήση των “greeklish” όπου ο υπολογιστής κάνει αυτόματη αντιστοίχιση των λέξεων αυτών στις αντίστοιχες ελληνικές προκειμένου να εμφανίσει τα αποτελέσματα που αναζητά ο χρήστης. Με αυτή ακριβώς την εφαρμογή θα ασχοληθούμε στην επόμενη ενότητα που θα λύσουμε ένα παράδειγμα σε γλώσσα Python.

3.4.1. ΠΑΡΑΔΕΙΓΜΑ ΣΕ PYTHON

Έχουμε ένα λεξικό 158 Αρχαίων Ελληνικών ονομάτων (Παράρτημα Α) γραμμένα σε "greeklish" και σε ελληνικά. Με εφαρμογή του αλγορίθμου που παρουσιάσαμε στην ενότητα 3.2., θέλουμε όταν ο χρήστης καταχωρεί το όνομα Χ σε "greeklish" να επιστρέφεται το ελληνικό όνομα με την ελάχιστη απόσταση Levenshtein.

Έστω ότι γίνεται η καταχώρηση του ονόματος "nepheli". Ο κώδικας εκτελεί τον αλγόριθμο για κάθε ένα όνομα του λεξικού και επιστρέφει μία προς μία τις αποστάσεις Levenshtein. Ταξινομώντας τις αποστάσεις από τη μικρότερη στη μεγαλύτερη και στη συνέχεια αντιστοιχίζοντας τις αποστάσεις με τα ελληνικά ονόματα του λεξικού, τα αποτελέσματα που εμφανίζουν τη μικρότερη απόσταση, όπως προκύπτουν από τον κώδικα (Παράρτημα Α) είναι τα εξής:

ΑΡΧΙΚΟ ΟΝΟΜΑ ΠΡΟΣ ΜΕΤΑΤΡΟΠΗ		ΟΝΟΜΑ (GREEKLISH)	ΑΠΟΣΤΑΣΗ ΚΑΤΆ LEVENSHTEIN		ΟΝΟΜΑ (ΕΛΛΗΝΙΚΑ)	ΑΠΟΣΤΑΣΗ ΚΑΤΆ LEVENSHTEIN		ΤΕΛΙΚΟ ΟΝΟΜΑ (ΕΛΛΗΝΙΚΑ)
nepheli	→	nefeli	2	→	Νεφέλη	2	→	Νεφέλη
		ekali	4		Εκάλη	4		
		neoklis	4		Νεοκλής	4		

Πίνακας 3.4.1: Αποτελέσματα μετατροπής ονόματος "nepheli" σε όνομα «Νεφέλη».

Δηλαδή το όνομα "nepheli" μετατρέπεται μέσω 2 αλλαγών στο όνομα "nefeli" και μέσω 4 αλλαγών στα ονόματα "ekali" και "neoklis", αντίστοιχα. Το όνομα "nefeli" επιλέγεται ως το όνομα με τη μικρότερη απόσταση Levenshtein, το οποίο με βάση το λεξικό αντιστοιχεί στο ελληνικό όνομα Νεφέλη.

n	e	p	h	e	l	i
|	|	|	|	|	|	|
n	e	f	-	e	l	i

Νεφέλη

Πίνακας 3.4.2: Διαδικασία μετατροπών (με κόκκινο χρώμα οι 2 αλλαγές)

Χρειάστηκε δηλαδή μία αντικατάσταση (R) και μία διαγραφή (D).

Αντίστοιχα αν καταχωρήσουμε το όνομα "leonidas" τα αποτελέσματα που παίρνουμε είναι τα εξής:

ΑΡΧΙΚΟ ΟΝΟΜΑ ΠΡΟΣ ΜΕΤΑΤΡΟΠΗ		ΟΝΟΜΑ (GREEKLISH)	ΑΠΟΣΤΑΣΗ ΚΑΤΆ LEVENSHTEIN		ΟΝΟΜΑ (ΕΛΛΗΝΙΚΑ)	ΑΠΟΣΤΑΣΗ ΚΑΤΆ LEVENSHTEIN		ΤΕΛΙΚΟ ΟΝΟΜΑ (ΕΛΛΗΝΙΚΑ)
leonidas	→	lewnidas	1	→	Λεωνίδας	1	→	Λεωνίδας
		leia	4		Λεία	4		
		ameinias	4		Αμεινίας	4		
		leandros	4		Λέανδρος	4		

Πίνακας 3.4.3: *Αποτελέσματα μετατροπής ονόματος "leonidas" σε όνομα «Λεωνίδας»*.

Τα προτεινόμενα ονόματα με το μικρότερο αριθμό αλλαγών είναι ο "lewnidas" με edit distance ίσο με 1 και οι "leia", "ameinias", "leandros" με edit distance ίσο με 4 το καθένα. Το όνομα "lewnidas" επιλέγεται ως το όνομα με τη μικρότερη απόσταση Levenshtein, το οποίο με βάση το λεξικό αντιστοιχεί στο ελληνικό όνομα Λεωνίδας.

l	e	o	n	i	d	a	s
|	|	|	|	|	|	|	|
l	e	w	n	i	d	a	s

↓

Λεωνίδας

Πίνακας 3.4.4: διαδικασία μετατροπών (με κόκκινο χρώμα η 1 αλλαγή)

Χρειάστηκε δηλαδή μόνο μία αντικατάσταση (R).

ΚΕΦΑΛΑΙΟ 4

4.1. ΠΡΟΒΛΗΜΑ ΕΛΑΧΙΣΤΗΣ ΔΙΑΔΡΟΜΗΣ (SHORTEST PATH PROBLEM)

Το Πρόβλημα Ελάχιστης Διαδρομής (Shortest Path - SP) είναι μία από τα πιο γνωστές εφαρμογές τόσο του δυναμικού προγραμματισμού όσο και της θεωρίας δικτύων [3]. Στόχος είναι η εύρεση ελάχιστης διαδρομής ξεκινώντας από ένα αρχικό σημείο και φτάνοντας σε ένα τελικό σημείο του δικτύου, μέσω πολλών ενδιάμεσων πιθανών διαδρομών. Αν και το όνομα του ίσως παραπέμπει αρχικά σε εύρεση βέλτιστων διαδρομών οδικών δικτύων μπορούμε να συναντήσουμε εφαρμογές αυτού του προβλήματος στα μέσα κοινωνικής δικτύωσης, σε τηλεφωνικά δίκτυα, τη ρομποτική κ.α. Μέσω της κατηγοριοποίησης της ενότητας 2.3.1 είναι προφανές ότι θα το κατατάξουμε σε ένα πρόβλημα ελαχιστοποίησης όπου τόσο οι καταστάσεις όσο και οι μεταβλητές απόφασης παρουσιάζονται από διακριτές μεταβλητές. Χαρακτηρίζεται ως ένα πρόβλημα κλάσης P δηλαδή επιλύεται σε πολυωνυμικό χρόνο από έναν ντετερμινιστικό αλγόριθμο. Στο παρόν κεφάλαιο θα αναλύσουμε τους τρόπους με τους οποίους μπορεί να λυθεί ένα τέτοιο πρόβλημα παρουσιάζοντας τόσο απλές εφαρμογές όσο και παραδείγματα βασισμένα σε πραγματικά δεδομένα.

4.2. ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ

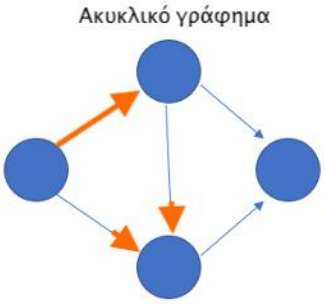
Το SP αναπαρίσταται από ένα γράφημα/δίκτυο (graph) {[3], [10]} το οποίο αποτελείται από κόμβους (vertices/ nodes) οι οποίοι συμβολίζουν την κατάσταση στην οποία βρισκόμαστε και ακμές (edges) που δηλώνουν τη μετάβαση από έναν κόμβο σε έναν άλλον. Σε κάθε μία ακμή δηλώνεται κάποιο κόστος (weight) το οποίο μπορεί να είναι απόσταση, χρόνος, χρηματικό κόστος κ.α. με το οποίο θα γίνει αυτή η μετάβαση. Το γράφημα αυτό μπορεί να είναι κατευθυνόμενο (directed) ή μη κατευθυνόμενο (undirected) [17]. Υποδηλώνεται, δηλαδή στο γράφημα η κατεύθυνση με την οποία γίνεται η μετάβαση από ένα σημείο Α σε ένα σημείο Β. Τα μη κατευθυνόμενα γραφήματα, είναι διπλής κατεύθυνσης (bidirectional) με την έννοια ότι αν μπορεί να γίνει η μετάβαση από ένα σημείο Α σε ένα σημείο Β τότε θα μπορεί να γίνει και η αντίστροφη διαδρομή, από το Β στο Α. Ακόμα ένας βασικός διαχωρισμός είναι τα κυκλικά (cyclic) και ακυκλικά (acyclic) γραφήματα [16]. Ένα μη προσανατολισμένο γράφημα θα είναι πάντα κυκλικό, με την έννοια ότι μπορούμε από έναν κόμβο Α να επιστρέψουμε στον ίδιο κόμβο μέσω μίας προσανατολισμένης διαδρομής. Αυτό στα ακυκλικά γραφήματα δεν γίνεται. Ένα ευθύ γράφημα μπορεί να είναι είτε κυκλικό (σχηματίζει τουλάχιστον έναν κύκλο) είτε μη κυκλικό.
Συνοπτικά , λοιπόν, έχουμε τους εξής διαχωρισμούς:

I. Με βάση την κατεύθυνση
    a. Κατευθυνόμενο
    b. Μη κατευθυνόμενο

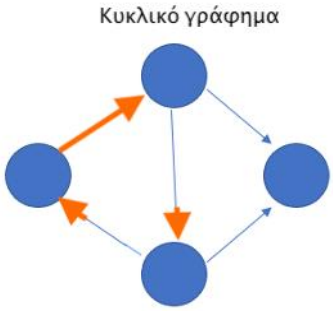
II. Με βάση την κυκλικότητα της διαδρομής
    a. Κυκλικό
    b. Ακυκλικό

Εικόνα 4.2.1.

Εικόνα 4.2.2.



Εικόνα 4.2.3.



Εικόνα 4.2.4.

Υπάρχουν πολλοί μέθοδοι επίλυσης του προβλήματος ελάχιστης διαδρομής. Εμείς θα επικεντρωθούμε σε δύο από αυτούς.

Α. Προς τα μπροστά αλγόριθμος, γνωστός ως Dijkstra Algorithm
Β. Προς τα πίσω μέθοδος πολλών σταδίων

4.3. ΑΛΓΟΡΙΘΜΟΣ DIJKSTRA

Ο αλγόριθμος αυτός είναι μια επαναληπτική μέθοδος για να βρούμε την ελάχιστη διαδρομή σε ένα γράφημα [18]. Όπως είναι γνωστό ο αλγόριθμος Dijkstra είναι αντικείμενο της Θεωρίας Δικτύων, η οποία είναι εκτός της παρούσας μελέτης, ωστόσο θα αναλυθεί στο παρόν κεφάλαιο ώστε να δώσουμε μία συνολική εικόνα ενός τόσο ευρέως διαδεδομένου προβλήματος καθώς μπορεί να επιλύσει ένα γράφημα όσο περίπλοκο κι αν είναι. Ο μόνος περιορισμός είναι ότι το κόστος που δηλώνεται στις ακμές δεν μπορεί να είναι αρνητικό.

Έχουμε πει ότι βασικό χαρακτηριστικό των μεθόδων Δυναμικού προγραμματισμού είναι η διάσπαση του προβλήματος σε υπό-προβλήματα και μέσω αναδρομικών σχέσεων που συνήθως λύνονται οπισθοδρομικά καταλήγουμε στη βέλτιστη λύση του γενικού προβλήματος. Αντίθετα, σε αυτή την περίπτωση, το βασικό χαρακτηριστικό του αλγόριθμου είναι ότι λύνεται με την προς τα εμπρός μέθοδο. Αυτό σημαίνει ότι προχωρώντας προοδευτικά βρίσκει την ελάχιστη διαδρομή για τη μετάβαση από έναν κόμβο στον επόμενο

και σταματάει όταν φτάσουμε στην τελική κατάσταση/προορισμό. Συμπεραίνουμε, λοιπόν, ότι η μέθοδος αυτή δεν λαμβάνει καθόλου υπόψιν τις αποφάσεις των προηγούμενων σταδίων, αντίθετα, επιλέγει τοπικά βέλτιστα. Με άλλα λόγια, παίρνει αποφάσεις με μόνο κριτήριο την κατάσταση την οποία βρίσκεται εκείνη τη στιγμή με την προοπτική ότι θα φτάσει τελικά σε κάποιο ολικό βέλτιστο. Αυτού του είδους οι αλγόριθμοι είναι οι λεγόμενοι greedy algorithms [19]. Ωστόσο, αν μπορούμε να βρούμε κάποια σύνδεση μεταξύ αυτών των αλγορίθμων και αυτών των αμιγώς δυναμικού προγραμματισμού είναι ότι και στις δύο περιπτώσεις ορίζεται μία αναδρομική σχέση και χωρίζεται το πρόβλημα σε μικρότερα υπό-προβλήματα.

4.3.1. ΠΕΡΙΓΡΑΦΗ ΑΛΓΟΡΙΘΜΟΥ DIJKSTRA

Ορίζουμε ως βάση ή πηγή (source) την αρχική κατάσταση και ως στόχο (target) την τελική κατάσταση.

Προκειμένου να ξεκινήσει ο αλγόριθμος χωρίζουμε τους κόμβους σε λυμένους και άλυτους. Λυμένοι χαρακτηρίζονται αυτοί οι οποίοι έχουμε ήδη επισκεφτεί και άλυτοι αυτοί που δεν έχουμε επισκεφτεί ακόμα. Οι κόμβοι συμβολίζονται συνήθως με γράμματα (A, B, C κ.ο.κ ή με αριθμούς 0, 1, 2, …). Ο αλγόριθμος Dijkstra ακολουθεί την παρακάτω μεθοδολογία [18]:

1. Στο πρώτο στάδιο θεωρούμε όλους τους κόμβους άλυτους εκτός από τον πρώτο (κόμβος πηγή) και ορίζουμε το κόστος του, δηλαδή την απόσταση από τον εαυτό του, να είναι μηδενικό.
2. Βρίσκουμε όλους τους γειτονικούς (δηλ. αυτούς που συνδέονται άμεσα με τον τρέχον κόμβο) άλυτους κόμβους και για κάθε έναν υπολογίζουμε τις υποψήφιες αποστάσεις που τους συνδέουν με τον τρέχοντα λυμένο κόμβο.

   Υποψήφια απόσταση = (απόσταση του λυμένου κόμβου από την πηγή) + (απόσταση που συνδέει απευθείας το λυμένο κόμβο με το γειτονικό άλυτο κόμβο)

   Ο αριθμός των γειτονικών άλυτων κόμβων καθορίζει και τον αριθμό των υποψήφιων αποστάσεων. Δηλαδή, οι υποψήφιες αποστάσεις μπορεί να είναι περισσότερες από μία γιατί μπορεί ο κόμβος να συνδέεται απευθείας με πολλούς άλυτους κόμβους.

3. Επιλέγουμε τη μικρότερη απόσταση από τις υποψήφιες και θεωρούμε πλέον λυμένο τον κόμβο εκείνον που συνδέεται με αυτή την ελάχιστη απόσταση.
   a. Η απόσταση αυτού του κόμβου ισούται πλέον με την επιλεχθείσα υποψήφια απόσταση
4. Αν ο νέος λυμένος κόμβος δεν είναι ο τελικός κόμβος τότε επαναλαμβάνουμε τα βήματα 2-3 μέχρι να βρεθούμε σε αυτόν.
   Αλλιώς, ο αλγόριθμος σταματάει.

   Τελικά έχουμε:

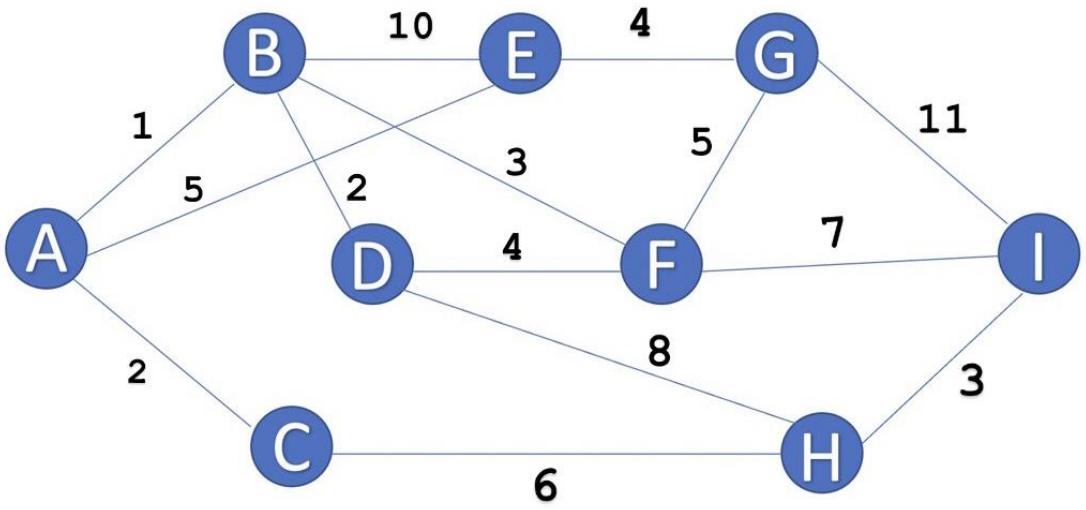
   Α. Τη συνολική ελάχιστη απόσταση της πιο σύντομης διαδρομής από την πηγή μέχρι τον τελικό κόμβο.

Β. Την ελάχιστη διαδρομή αρχίζοντας από τον τελικό προορισμό προς την αρχή, αποτελούμενη από τις ακμές που έχουν θεωρηθεί ότι οδηγούν στη βέλτιστη διαδρομή.

4.3.2. ΠΑΡΑΔΕΙΓΜΑ

Μπορούμε λοιπόν να εφαρμόσουμε την παραπάνω μέθοδο σε ένα παράδειγμα.

Έστω ότι έχουμε το παρακάτω δίκτυο.



Σχήμα 4.3.1: Γραφική αναπαράσταση μη κατευθυνόμενου δικτύου

Για τη λύση του προβλήματος δημιουργούμε έναν πίνακα ο οποίος παρουσιάζει την κατάσταση του προβλήματος σε κάθε βήμα.

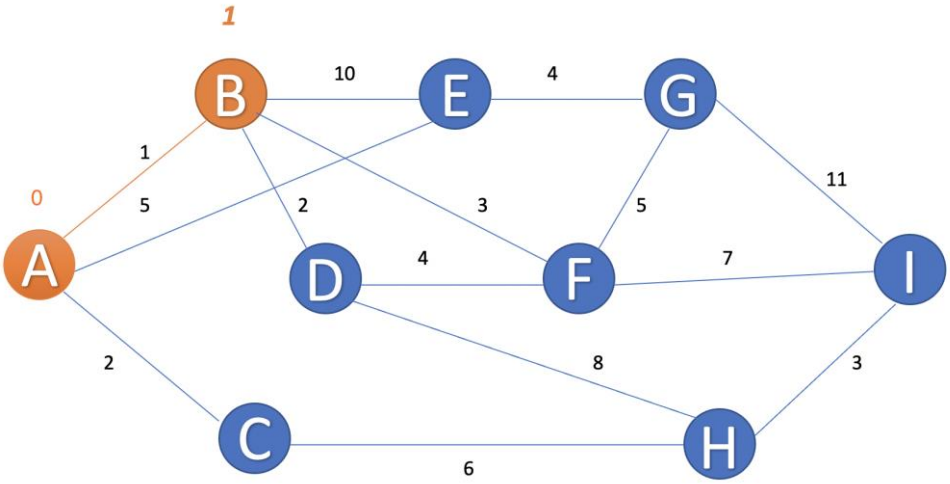
ΣΥΝΟΛΟ ΛΥΜΕΝΩΝ ΚΟΜΒΩΝ	ΓΕΙΤΟΝΙΚΟΙ ΚΟΜΒΟΙ	ΥΠΟΨΗΦΙΑ ΑΠΟΣΤΑΣΗ	ΝΕΟΣ ΛΥΜΕΝΟΣ ΚΟΜΒΟΣ	ΕΛΑΧΙΣΤΗ ΑΠΟΣΤΑΣΗ
{A}	A--> B	1	B	1
	A--> C	2		
	A--> E	5		
{A,B}	A-->C	2	C	2
	A--> E	5		
	B--> D	1+2=3		
	B--> E	1+10=11		
	B--> F	1+3= 4		
{A,B,C}	A--> E	5	D	3
	B--> D	3		
	B--> E	11		
	B--> F	4		
	C--> H	2+6=8		
{A,B,C,D}	A--> E	5	F	4
	B--> E	11		
	B--> F	4		
	C--> H	8		
	D--> F	3+4=7		
	D--> H	3+8=11		
{A,B,C,D,F}	A--> E	5	E	5
	B--> E	11		
	C--> H	8		
	D--> H	11		
	F--> G	4+5=9		
	F--> I	4+7=11		
{A,B,C,D,F,E}	C--> H	8	H	8
	D--> H	11		
	E--> G	4+5=9		
	F--> G	9		
	F--> I	11		
{A,B,C,D,F,E,H}	E--> G	5+4=9	G	9
	F--> G	9		
	F--> I	11		
	H--> I	11		
{A,B,C,D,F,E,H,G}	G--> I	9+11=20	I	11
	F--> I	11		
	H--> I	8+3=11		

Πίνακας 4.3.1: Υπολογισμοί του αλγόριθμου Dijkstra

ΑΝΑΛΥΣΗ ΠΙΝΑΚΑ

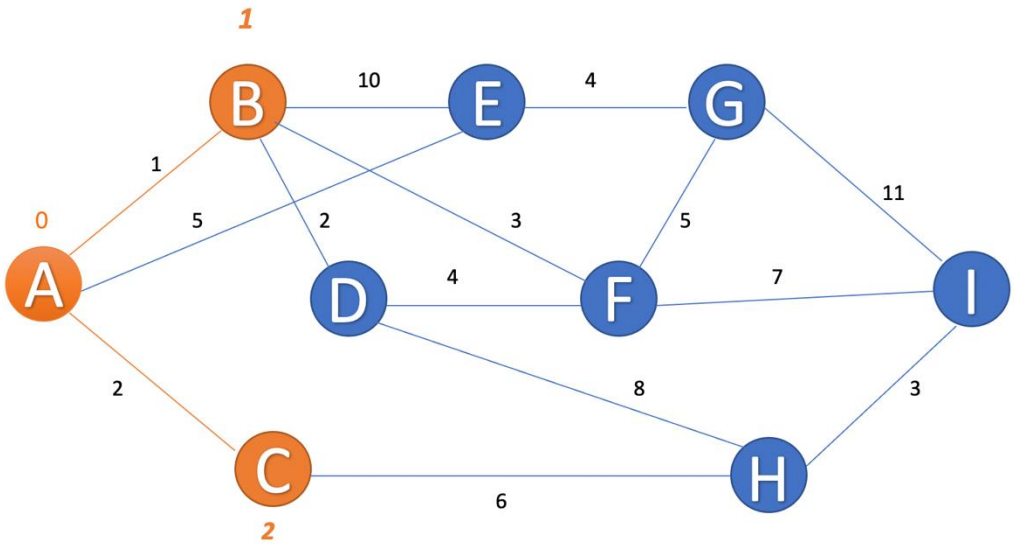
Ο πίνακας αποτελείται από 5 στήλες. Στην πρώτη στήλη αναγράφονται οι λυμένοι κόμβοι, αυτοί δηλαδή που έχουμε επισκεφτεί.

Στο *1ο* στάδιο ο μόνος λυμένος κόμβος είναι η πηγή {A} στην οποία ορίζουμε η απόσταση από τον εαυτό της να είναι 0. Όπως βλέπουμε το A συνδέεται άμεσα με τους άλυτους κόμβους B, C & E. Οι υποψήφιες αποστάσεις είναι 1 ,2 και 5, αντίστοιχα. Συνεπώς η ελάχιστη απόσταση από την πηγή είναι ίση με 1. Ορίζουμε η προσωρινή ελάχιστη απόσταση από την πηγή να είναι ίση με 1 και στο επόμενο βήμα θεωρούμε ότι ο κόμβος B είναι λυμένος.
Το δίκτυο δηλαδή βρίσκεται σε αυτή την κατάσταση:



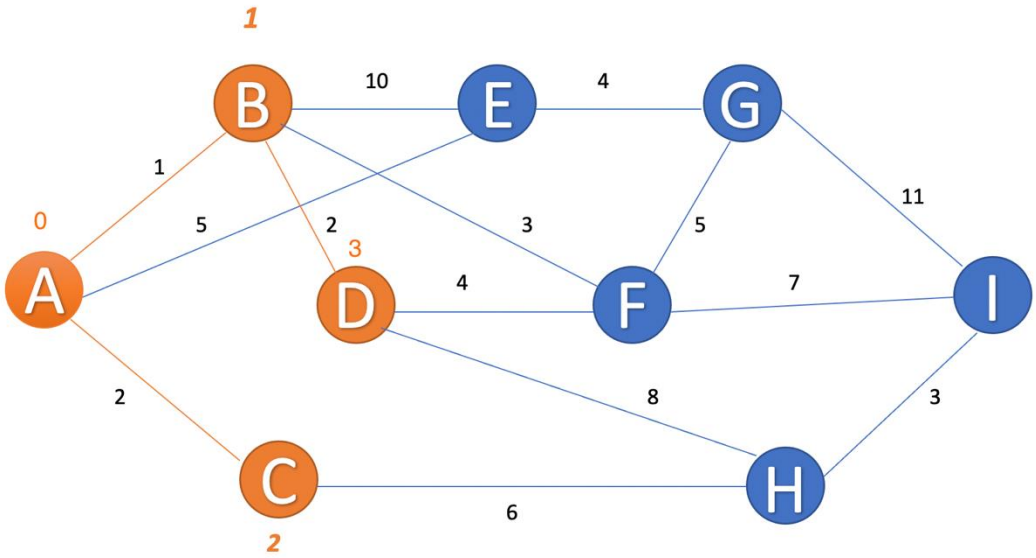
Σχήμα 4.3.2: Κατάσταση δικτύου όταν A & B λυμένοι κόμβοι

Στο 2ο στάδιο έχουμε τους λυμένους κόμβους {A, B} οι οποίοι έχουν ως υποψήφιες αποστάσεις τις AC, AE, BD, BE & BF όπως ακριβώς φαίνεται στο 2ο βήμα του πίνακα. Εδώ ο πιο κοντινός κόμβος είναι ο C από τον A με απόσταση 2. Έτσι, στο επόμενο βήμα ο κόμβος C θεωρείται λυμένος. Η διαδρομή μέχρι αυτό το σημείο έχει την παρακάτω μορφή:



Σχήμα 4.3.3: Κατάσταση δικτύου όταν A, B & C λυμένοι κόμβοι

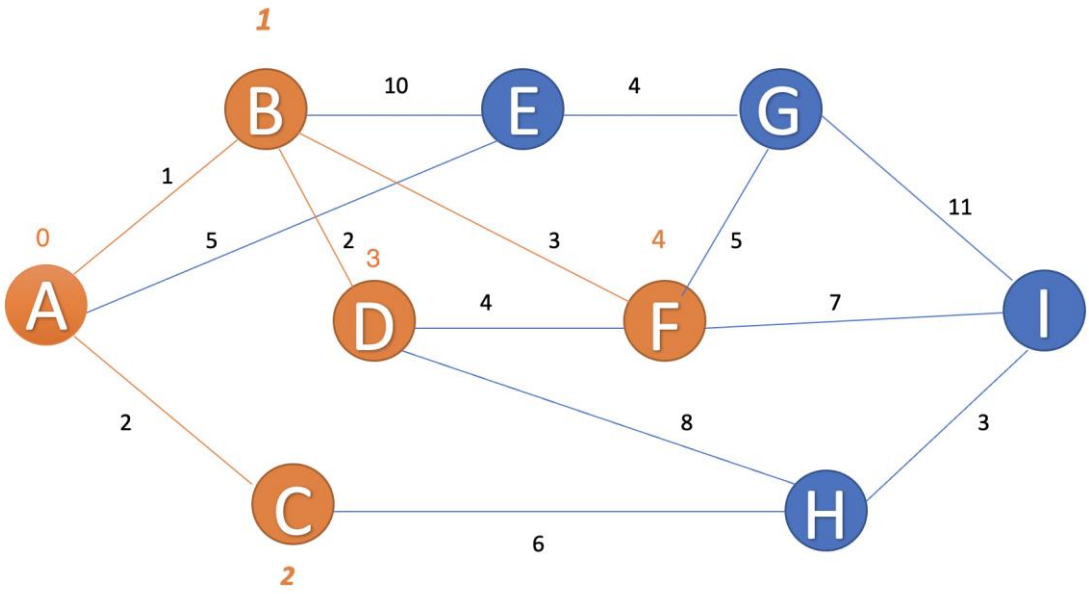
Στο 3ο στάδιο έχουμε τους λυμένους κόμβους {A, B, C} οι οποίοι έχουν ως υποψήφιες αποστάσεις τις AE, BD, BE, BF & CH όπως ακριβώς φαίνεται στο 3ο βήμα του πίνακα. Εδώ ο πιο κοντινός κόμβος είναι ο D από τον B με απόσταση 1+2=3. Έτσι, στο επόμενο βήμα ο κόμβος D θεωρείται λυμένος. Η διαδρομή μέχρι αυτό το σημείο έχει την παρακάτω μορφή:



Σχήμα 4.3.4: Κατάσταση δικτύου όταν A, B, C & D λυμένοι κόμβοι.

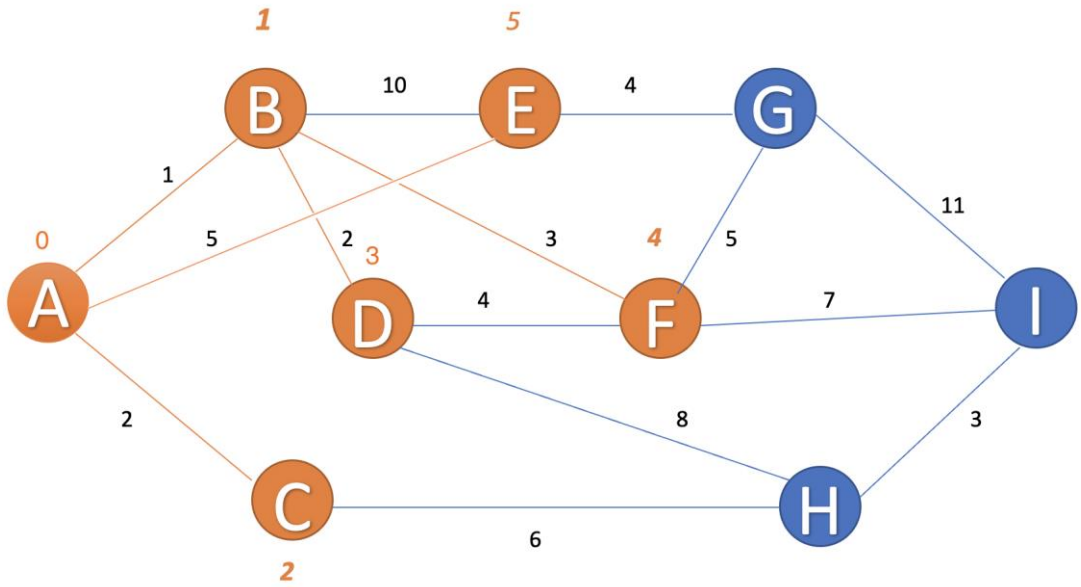
Με ακριβώς ανάλογο τρόπο συνεχίζουμε να υπολογίζουμε τους υπόλοιπους άλυτους κόμβους και στα παρακάτω σχήματα παρουσιάζεται γραφικά η διαδρομή σε κάθε βήμα μέχρι να φτάσουμε στον τελικό προορισμό (κόμβος I).

Μικρότερη υποψήφια απόσταση B→ F ίση με 4



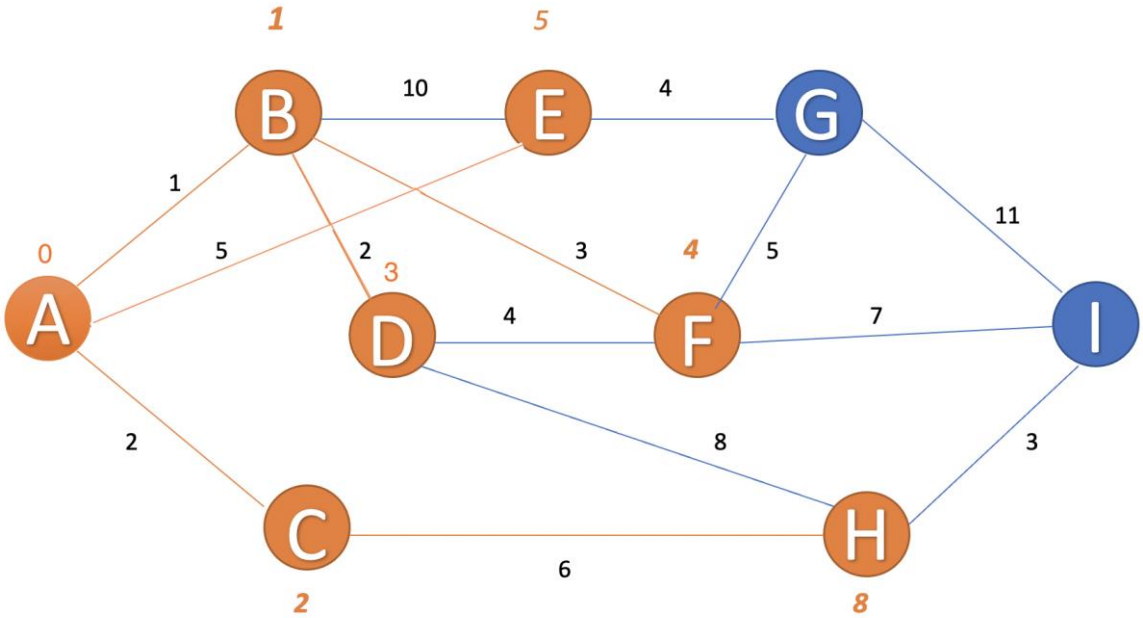
Σχήμα 4.3.5: Κατάσταση δικτύου όταν A, B, C, D & F λυμένοι κόμβοι.

Μικρότερη υποψήφια απόσταση A→ E ίση με 5



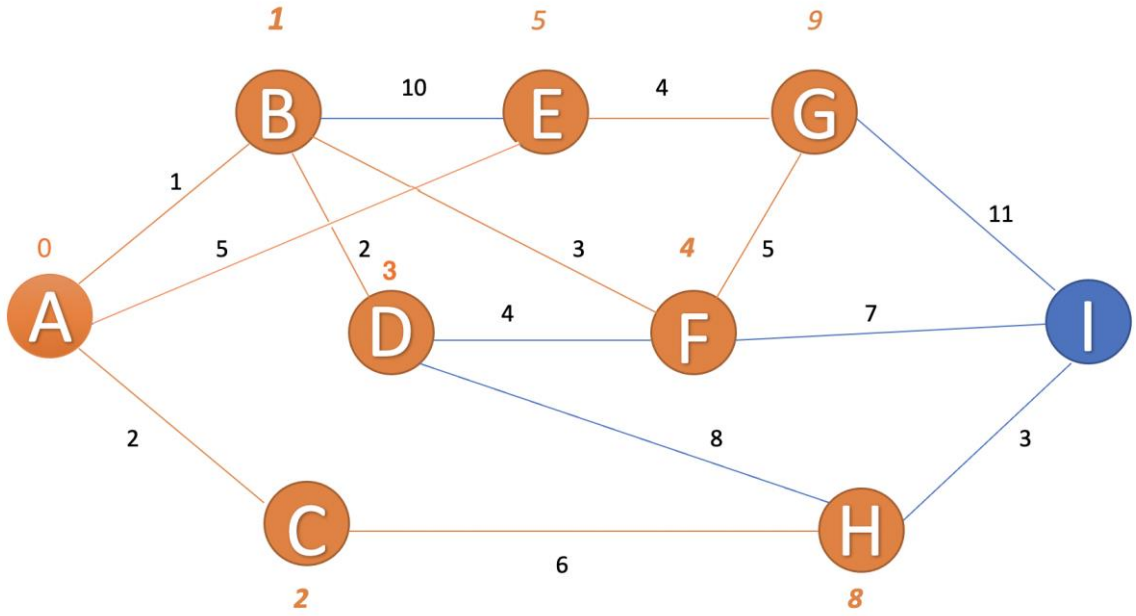
Σχήμα 4.3.6: Κατάσταση δικτύου όταν A, B, C, D, F & E λυμένοι κόμβοι

Μικρότερη υποψήφια απόσταση C→ H ίση με 8



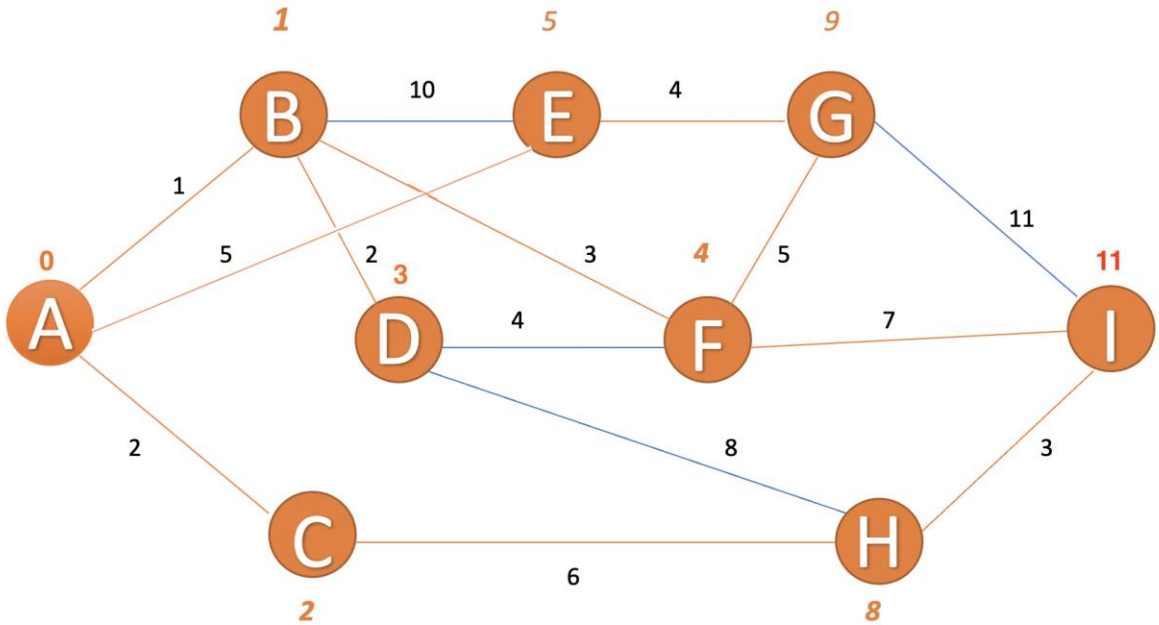
Σχήμα 4.3.7: Κατάσταση δικτύου όταν A, B, C, D, F, E & H λυμένοι κόμβοι

Μικρότερη υποψήφια απόσταση E→ G και F→G ίση με 9



Σχήμα 4.3.8: Κατάσταση δικτύου όταν A, B, C, D, F ,E ,H & G λυμένοι κόμβοι.

Μικρότερη υποψήφια απόσταση F→ I και H→ I ίση με 11



Σχήμα 4.3.9: Κατάσταση δικτύου όταν όλοι οι κόμβοι έχουν επισκεφθεί.

Εφόσον φτάσαμε στον τελικό κόμβο I ο αλγόριθμος σταματάει.

Βλέποντας τώρα από το τέλος προς την αρχή τη βέλτιστη διαδρομή έχουμε τις εξής περιπτώσεις:

I → H → C → A ή
I → F → B → A

Τέλος, τα φέρνουμε στην κανονική μορφή ξεκινώντας από την πηγή A μέχρι τον προορισμό I όπου έχουμε δύο βέλτιστες διαδρομές

A → C → H → I ή A → B → F → I με συνολικό κόστος (απόσταση) 11.

4.4. ΟΠΙΣΘΟΔΡΟΜΙΚΗ ΜΕΘΟΔΟΣ ΔΥΝΑΜΙΚΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Όπως αναφέραμε παραπάνω ένας άλλος τρόπος προσέγγισης του προβλήματος ελάχιστης διαδρομής είναι μέσω της οπισθοδρομικής αναδρομής (Backwards Recursion). Χωρίζουμε το πρόβλημα σε m στάδια και κάνοντας βήματα από πίσω προς τα εμπρός καταλήγουμε στη βέλτιστη διαδρομή του συνολικού προβλήματος. Ομοίως με την παραπάνω μέθοδο ορίζουμε κάθε κόμβο να είναι μία κατάσταση i και κάθε ακμή να είναι το κόστος f_{ij} για τη μετάβαση από μια κατάσταση i ενός σταδίου m στην κατάσταση j του επόμενου σταδίου m+1. Ορίζουμε την εξής αναδρομική σχέση [3]:

$$F_m(i) = \min_j \{ f_{ij} + F_{m+1}(j) \} \qquad m=1,2,\dots,K-1 \qquad (4.1)$$

Όπου $F_m(i)$ είναι το αθροιστικό κόστος για τη μετάβαση από την κατάσταση i του σταδίου m μέχρι την τελική κατάσταση του τελευταίου σταδίου. Στη συγκεκριμένη περίπτωση, σε αντίθεση με τον αλγόριθμο Dijkstra μπορούμε να θεωρήσουμε ως βάση τον τελικό κόμβο όπου θέτουμε την απόσταση ίση με 0 και ως στόχο την κατάσταση του 1ου σταδίου όπου και θα σταματήσει η μέθοδος. Παρακάτω θα παρουσιάσουμε ένα πρόβλημα ακυκλικού, κατευθυνόμενου δικτύου.

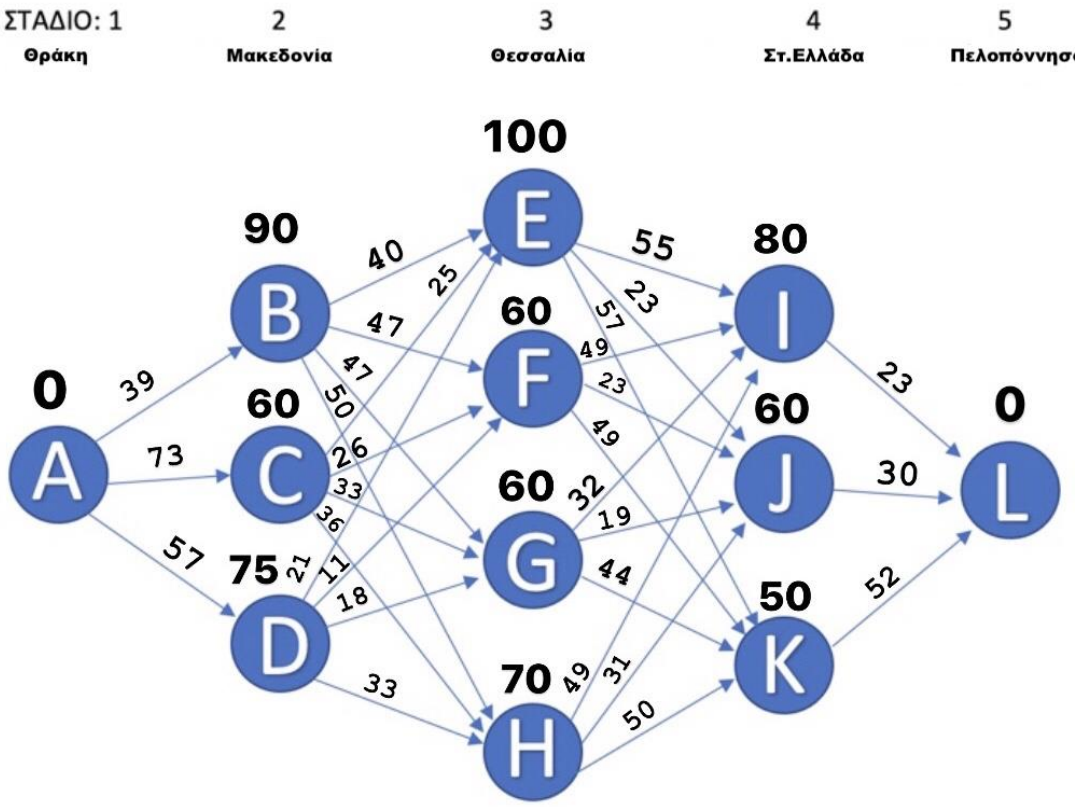
4.4.1. ΠΑΡΑΔΕΙΓΜΑ

Μία παρέα φοιτητών που σπουδάζουν στην Αλεξανδρούπολη οργανώνουν ένα ταξίδι οδικώς στην Ελλάδα με τελικό προορισμό τα σπίτια τους στην Πάτρα. Ενδιάμεσα θέλουν να διασχίσουν 3 γεωγραφικά διαμερίσματα (Μακεδονία, Θεσσαλία, Στερεά Ελλάδα) διανυκτερεύοντας σε ξενοδοχείο κάποιας πόλης που ανήκει στο εκάστοτε γεωγραφικό διαμέρισμα. Στόχος των παιδιών είναι να βρουν το οικονομικότερο πακέτο για το ταξίδι τους, δηλαδή την ελάχιστη διαδρομή για κάθε πόλη ώστε να έχουν το ελάχιστο κόστος σε βενζίνες καθώς και το ελάχιστο κόστος διαμονής. Στον παρακάτω πίνακα φαίνονται τα συνολικά κόστη της βενζίνης[6] και τα συνολικά κόστη των ξενοδοχείων για τη διαμονής τους αναλόγως την πόλη που θα επιλέξουν να διανυκτερεύσουν.

[6] Όπως υπολογίστηκαν από τη σελίδα "Εύρεση Απόστασης-Διαδρομής Μεταξύ Πόλεων, Περιοχών Στο Χάρτη. Υπολογισμός Κόστους Ταξιδιού (Καύσιμα, Διόδια)." *Vriskoapostasi.gr*, vriskoapostasi.gr/el/.

				ΚΟΣΤΟΣ ΒΕΝΖΙΝΗΣ (€)												
				ΠΡΟΟΡΙΣΜΟΣ												
	ΓΕΩΓΡΑΦΙΚΟ ΔΙΑΜΕΡΙΣΜΑ			ΘΡΑΚΗ	ΜΑΚΕΔΟΝΙΑ			ΘΕΣΣΑΛΙΑ				ΣΤ.ΕΛΛΑΔΑ			ΠΕΛΟΠΟΝΝΗΣΟΣ	
			ΠΟΛΕΙΣ	Αλεξανδρούπολη	Σέρρες	Φλώρινα	Γρεβενά	Λάρισα	Τρίκαλα	Καρδίτσα	Βόλος	Μεσολόγγι	Καρπενήσι	Χαλκίδα	Πάτρα	
ΠΡΟΕΛΕΥΣΗ	ΘΡΑΚΗ	A	Αλεξανδρούπολη	-	39	73	57	-	-	-	-	-	-	-	-	
	ΜΑΚΕΔΟΝΙΑ	B	Σέρρες	-	-	-	-	40	47	47	50	-	-	-	-	
		C	Φλώρινα	-	-	-	-	25	26	33	36	-	-	-	-	
		D	Γρεβενά	-	-	-	-	21	11	18	33	-	-	-	-	
	ΘΕΣΣΑΛΙΑ	E	Λάρισα	-	-	-	-	-	-	-	-	55	23	57	-	
		F	Τρίκαλα	-	-	-	-	-	-	-	-	49	23	49	-	
		G	Καρδίτσα	-	-	-	-	-	-	-	-	32	19	44	-	
		H	Βόλος	-	-	-	-	-	-	-	-	49	31	50	-	
	ΣΤ.ΕΛΛΑΔΑ	I	Μεσολόγγι	-	-	-	-	-	-	-	-	-	-	-	23	
		J	Καρπενήσι	-	-	-	-	-	-	-	-	-	-	-	30	
		K	Χαλκίδα	-	-	-	-	-	-	-	-	-	-	-	52	
	ΠΕΛΟΠΟΝΝΗΣΟΣ	L	Πάτρα	-	-	-	-	-	-	-	-	-	-	-	-	
ΚΟΣΤΟΣ ΔΙΑΜΟΝΗΣ (€)				-	90	60	75	100	60	60	70	80	60	50	0	

Πίνακας 4.4.1 : Συνολικά κόστη βενζίνης και διαμονής για κάθε πόλη



Σχήμα 4.4.1: Γραφική αναπαράσταση του δικτύου (5 στάδια, 12 καταστάσεις)

Το πρόβλημα, χωρίζεται σε 5 στάδια. Το κάθε στάδιο αντιστοιχεί στο γεωγραφικό διαμέρισμα που οι φοιτητές έχουν αποφασίσει να διανυκτερεύσουν. Κάθε κόμβος αντιστοιχεί στο ξενοδοχείο διαμονής σε κάθε μία πόλη και για λόγους ευκολίας θα ταυτίσουμε τα ξενοδοχεία αυτά με τις αντίστοιχες πόλεις στις οποίες βρίσκονται. Πάνω από κάθε κόμβο παρουσιάζεται το κόστος διαμονής κάθε ξενοδοχείου. Οι ακμές, δηλώνουν τη μετάβαση από μία πόλη ενός γεωγραφικού διαμερίσματος, σε μία πόλη του επόμενου γεωγραφικού διαμερίσματος. Το κόστος της βενζίνης παρουσιάζεται πάνω από κάθε ακμή.

Η αναδρομική σχέση του παραδείγματος διαμορφώνεται ως εξής :

$$F_m(i) = min\{ g_i + f_{ij} + F_{m+1}(j) \} \quad m=1,2,\ldots,K-1 \qquad (4.2)$$

όπου g_i= κόστος διαμονής της εκάστοτε κατάστασης i
f_{ij} = κόστος βενζίνης για τη μετάβαση από την πόλη i στην πόλη j

Ξεκινάμε τους υπολογισμούς για την οπισθοδρομική μέθοδο εφαρμόζοντας την αναδρομική σχέση (4.2).

4ο στάδιο

Υπολογίζουμε την ελάχιστη διαδρομή από κάθε κατάσταση του 4ου σταδίου προς την κατάσταση L του τελευταίου σταδίου 5. Γνωρίζουμε ότι το κόστος διαμονής στην κατάσταση L είναι 0, δηλαδή g_i=0 και εφόσον αυτή είναι η τελευταία κατάσταση δεν υπάρχει μετάβαση προς κάποια άλλη πόλη επόμενου σταδίου οπότε f_{ij}=0. Έτσι, έχουμε ότι $F_5(j)$= 0.
Εφαρμόζοντας τον τύπο (4.2) έχουμε τα εξής αποτελέσματα:

στάδιο 4						
καταστάσεις i	επόμενη κατάσταση j	κόστος διαμονής g_i	κόστος βενζίνης fij	$F_5(j)$	ελάχιστο κόστος ως τελική κατάσταση	ελάχιστη πρόσφατη σύνδεση
I	L	80	23	0	103	I --> L
J		60	30		90	J--> L
K		50	52		102	K--> L

Πίνακας 4.4.2: Υπολογισμοί 4ου σταδίου

3ο στάδιο

Σε αυτό το στάδιο υπάρχουν οι καταστάσεις E, F, G, H. Από την κατάσταση E (Λάρισα) μπορούμε να οδηγηθούμε στον τελικό προορισμό (Πάτρα) είτε μέσω της κατάστασης I (Μεσολόγγι), είτε μέσω της κατάστασης J (Καρπενήσι) είτε μέσω της κατάστασης K (Χαλκίδα). Από τον τύπο (4.2) έχουμε:

$$\mathrm{F3(E)} = \min_{j=I,J,K}\{ g(E) + f(E,j) + F4(j)\} = \min\{ g(E) + f(E,I) + F4(I), g(E) + f(E,J) + F4(J), g(E) +$$

$$f(E,K) + F4(K)\} = \min\{100 + 55 + 103, 100 + 23 + 90, 100 + 57 + 102\} = \min\{258, 213, 259\} = 213$$

Η βέλτιστη απόφαση όταν βρισκόμαστε στην κατάσταση E είναι η επιλογή της κατάστασης J του επόμενου σταδίου. Από τη Λάρισα δηλαδή, η βέλτιστη απόφαση είναι η μετάβαση και η διαμονή σε ξενοδοχείο του Καρπενησίου με ελάχιστο κόστος F_3(E) = 213.

Ομοίως, για την κατάσταση F παίρνουμε τα παρακάτω αποτελέσματα:

$$\mathrm{F3(F)} = \min_{j=I,J,K}\{ g(F) + f(F,j) + F4(j)\} = \min\{60 + 49 + 103, 60 + 23 + 90, 60 + 49 + 102\} =$$
$$= \min\{212, 173, 211\} = 173$$

Η βέλτιστη απόφαση από την κατάσταση F (Τρίκαλα) είναι η επιλογή της κατάστασης J (Καρπενήσι) του επόμενου σταδίου με τιμή $F_3(F) = 173$.

Για την κατάσταση G υπολογίζουμε ότι:

$$\mathrm{F3}(G) = \min_{j=I,J,K}\{\, g(G) + f(G,j) + F4(j)\} = \min\{60+32+103, 60+19+90, 60+44+102\} =$$

$$= \min\{195,169,206\} = 169$$

Αντίστοιχα, η βέλτιστη απόφαση βρισκόμενοι στην κατάσταση G (Καρδίτσα) είναι να ακολουθήσουμε τη διαδρομή που οδηγεί στην κατάσταση J (Καρπενήσι) του 4ου σταδίου. Το ελάχιστο κόστος διαδρομής είναι $F_3(G) = 169$.

Τέλος, για την κατάσταση H έχουμε αντιστοίχως τα εξής αποτελέσματα:

$$\mathrm{F3}(H) = \min_{j=I,J,K}\{\, g(H) + f(H,j) + F4(j)\} = \min\{70+49+103, 70+31+90, 70+50+102\} =$$

$$= \min\{222, 191, 222\} = 191$$

Συνεπώς, η βέλτιστη απόφαση όταν βρισκόμαστε στην τρέχουσα κατάσταση H (Βόλος) είναι να επιλέξουμε ως επόμενη κατάσταση την J του επόμενου σταδίου με ελάχιστο κόστος διαδρομής $F_3(H)= 191$.

Συνολικά οι υπολογισμοί του 3ου σταδίου φαίνονται στον παρακάτω πίνακα:

στάδιο 3						
καταστάσεις i	επόμενη κατάσταση j	κόστος διαμονής g_i	κόστος βενζίνης fij	$F_4(j)$	ελάχιστο κόστος ως τελική κατάσταση	ελάχιστη πρόσφατη σύνδεση
E	I	100	55	103	258	
	J		23	90	213	E--> J
	K		57	102	259	
F	I	60	49	103	212	
	J		23	90	173	F--> J
	K		49	102	211	
G	I	60	32	103	195	
	J		19	90	169	G--> J
	K		44	102	206	
H	I	70	49	103	222	
	J		31	90	191	H--> J
	K		50	102	222	

Πίνακας 4.4.3: Υπολογισμοί 3ου σταδίου

2ο στάδιο

Στο 2ο στάδιο υπάρχουν οι καταστάσεις B, C, D. Όμοια με τους υπολογισμούς των παραπάνω σταδίων παρουσιάζουμε τα αποτελέσματα στον παρακάτω πίνακα.

στάδιο 2						
καταστάσεις i	επόμενη κατάσταση j	κόστος διαμονής g_i	κόστος βενζίνης fij	$F_3(j)$	ελάχιστο κόστος ως τελική κατάσταση	ελάχιστη πρόσφατη σύνδεση
B	E	90	40	213	343	
	F		47	173	310	
	G		47	169	306	B --> G
	H		50	191	331	
C	E	60	25	213	298	
	F		26	173	259	C --> F
	G		33	169	262	
	H		36	191	287	
D	E	75	21	213	309	
	F		11	173	259	D --> F
	G		18	169	262	
	H		33	191	299	

Πίνακας 4.4.4: Υπολογισμοί 2ου σταδίου

Από την κατάσταση B (Σέρρες) η βέλτιστη απόφαση είναι να βρεθούμε στην κατάσταση G (Καρδίτσα) του επόμενου σταδίου με ελάχιστο κόστος $F_2(B) = 306$.

Από την κατάσταση C (Φλώρινα) η βέλτιστη απόφαση είναι η επιλογή της κατάστασης F (Τρίκαλα) του 3ου σταδίου με ελάχιστη τιμή $F_2(C) = 259$.

Από την κατάσταση D (Γρεβενά) η βέλτιστη απόφαση είναι να βρεθούμε στην κατάσταση F του 3ου σταδίου με ελάχιστο κόστος διαδρομής $F_2(D) = 259$.

1ο στάδιο

Αυτό είναι και το τελευταίο στάδιο που θα υπολογίσουμε, καθώς υπάρχει πλέον μόνο μία κατάσταση, η αρχική A (Αλεξανδρούπολη). Στον πίνακα παρουσιάζονται τα αποτελέσματα της μετάβασης από την κατάσταση A σε μία από τις τρεις πιθανές καταστάσεις (B, C & D) του επόμενου σταδίου.

στάδιο 1						
καταστάσεις i	επόμενη κατάσταση j	κόστος διαμονής g_i	κόστος βενζίνης fij	$F_2(j)$	ελάχιστο κόστος ως τελική κατάσταση	ελάχιστη πρόσφατη σύνδεση
A	B	0	39	306	345	
	C		73	259	332	
	D		57	259	316	A --> D

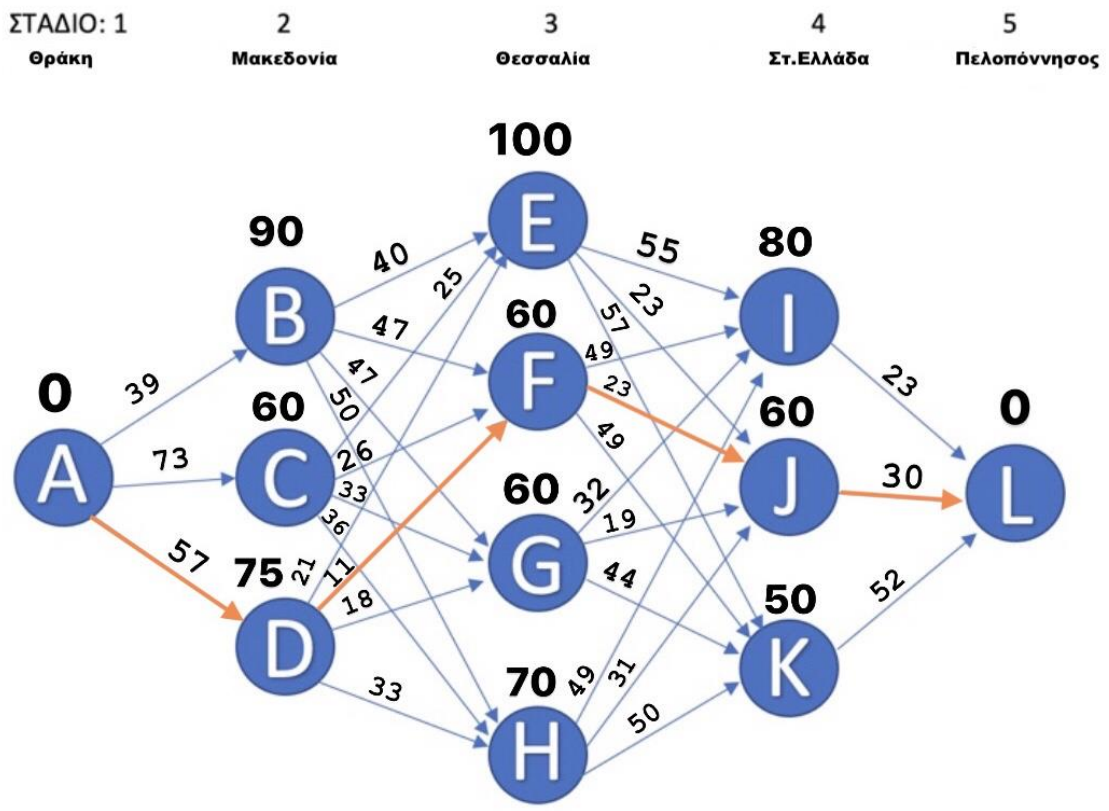
Πίνακας 4.4.5: Υπολογισμοί 1ου σταδίου

Όπως φαίνεται η βέλτιστη απόφαση όταν βρισκόμαστε στην αρχική κατάσταση είναι να επιλέξουμε την κατάσταση D (Γρεβενά) με ελάχιστο κόστος διαδρομής $F_1(C) = 316$.

ΑΝΤΙΣΤΡΟΦΗ ΔΙΑΔΡΟΜΗΣ:

Ξεκινώντας τώρα από την αρχική κατάσταση A επιλέγουμε ως επόμενη την κατάσταση D καθώς αυτή παρουσίαζε το μικρότερο κόστος (316) σε σχέση με τις άλλες υποψήφιες καταστάσεις B & C του 2ου σταδίου. Ενώ βρισκόμαστε στην κατάσταση D βλέπουμε ότι η βέλτιστη απόφαση είναι να βρεθούμε στην κατάσταση F του 3ου σταδίου με κόστος 259. Στη συνέχεια, βρισκόμενοι στην κατάσταση F επιλέγουμε ως βέλτιστη απόφαση να βρεθούμε στην κατάσταση J του επόμενου, 4ου σταδίου, αφού παρουσιάζει το μικρότερο κόστος (173) σε σχέση με τις εναλλακτικές καταστάσεις I, K. Τέλος, από την κατάσταση J φτάνουμε στον τελικό προορισμό, την κατάσταση L του τελευταίου σταδίου με ελάχιστο κόστος διαδρομής 90. Συνεπώς, η διαδρομή που οδηγεί στη βέλτιστη λύση είναι η A → D → F → J → L με συνολικό κόστος ίσο με 316.
Με άλλα λόγια η διαδρομή που πρέπει να ακολουθήσουν οι φοιτητές είναι Αλεξανδρούπολη → Γρεβενά → Τρίκαλα → Καρπενήσι → Πάτρα με συνολικό κόστος ίσο με 316.

Η βέλτιστη διαδρομή παρουσιάζεται στο παρακάτω σχήμα:



Σχήμα 4.4.2: Βέλτιστη διαδρομή

4.5. ΠΡΟΒΛΗΜΑ ΕΛΑΧΙΣΤΗΣ ΔΙΑΔΡΟΜΗΣ ΚΑΙ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

Ο όρος «Δυναμικός Προγραμματισμός», όπως αναφέρεται στην εισαγωγή, χρησιμοποιήθηκε για να δηλώσει από τη μία τον παράγοντα του χρόνου στα πολυβάθμια προβλήματα (Δυναμικός) και από την άλλη την εύρεση μεθόδων για τον βέλτιστο προγραμματισμό υπό την έννοια της οργάνωσης (Προγραμματισμός) [21]. Δηλαδή, δεν ονομάστηκε έτσι, ώστε να σχετίζεται με αυτό που ονομάζεται «επιστήμη των δεδομένων» (Data Science) με τη χρήση προγραμματιστικών γλωσσών. Ωστόσο, από τα πρώτα κιόλας βήματα της ανάπτυξης της τεχνολογίας, οι διάφορες γλώσσες προγραμματισμού αποτέλεσαν ένα ισχυρό εργαλείο για τη διαχείριση μεγάλων βάσεων δεδομένων με σκοπό να αναλυθούν και να επιλυθούν προβλήματα που η λύση τους με χαρτί και μολύβι φαντάζει αδύνατη. Θέλοντας λοιπόν να δώσουμε έμφαση στη σημαντικότητα της χρήσης του προγραμματισμού στον Δυναμικό Προγραμματισμό, θα παρουσιάσουμε ένα παράδειγμα βασισμένο σε πραγματικά δεδομένα με τη χρήση της Python.

4.5.1. ΠΑΡΑΔΕΙΓΜΑ

Στο συγκεκριμένο παράδειγμα παρουσιάζεται ένα μεγάλο κομμάτι του οδικού δικτύου, της πρωτεύουσας της Ιταλίας, της Ρώμης, από το 1999 [22]. Το δίκτυο αποτελείται από 3353 κόμβους και 8870 ακμές. Υλοποιείται μέσω κώδικα ο αλγόριθμος Dijkstra ώστε να βρεθεί η ελάχιστη διαδρομή από κάθε έναν κόμβο σε κάθε άλλον. Δεδομένου ότι τα δεδομένα είναι πραγματικά δεν υπάρχουν αρνητικές τιμές στις ακμές οπότε ο αλγόριθμος εφαρμόζεται κανονικά. Στόχος μας, είναι η εύρεση της ελάχιστης διαδρομής από τον κόμβο 1 έως τον τελευταίο κόμβο 3353.

Παρακάτω επεξηγούνται συνοπτικά όλοι οι βασικοί όροι [8] που συναντάμε στον κώδικα [8] ο οποίος βρίσκεται στο ΠΑΡΑΡΤΗΜΑ Α.

ΒΑΣΙΚΟΙ ΟΡΟΙ ΚΩΔΙΚΑ

Original_source: αρχικός κόμβος- πηγή. Έστω ότι είναι ο κόμβος 1 και δεν αλλάζει καθόλη τη διαδρομή.

Source_node: εκάστοτε κόμβος-πηγή ή αλλιώς, τρέχων κόμβος.

Adjacent_node: γειτονικός κόμβος, δηλαδή κάθε κόμβος που συνδέεται απευθείας με τον τρέχοντα κόμβο (source_node).

Distance[source_node]: η απόσταση από τον τρέχοντα κόμβο.

Length_to_adjacent_node_from_source: κόστος ακμής, δηλαδή η απόσταση για τη μετάβαση από τον τρέχοντα κόμβο-πηγή σε έναν γειτονικό κόμβο.

Distance[adjacent_node]: Length_to_adjacent_node_from_source + Distance[source_node] ⟺ μήκος γειτονικού κόμβου από τον τρέχοντα κόμβο + απόσταση τρέχοντα κόμβου από τον αρχικό κόμβο πηγή.

Destination_node: κάθε κόμβος μπορεί να θεωρηθεί ως τελικός κόμβος. Στο συγκεκριμένο παράδειγμα θεωρούμε τελικό κόμβο τον κόμβο 3353.

ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ ΤΗΣ PYTHON

DICTIONARY (ΛΕΞΙΚΟ):

Είναι μία συλλογή [24] που αποθηκεύει δεδομένα σε ζεύγη {key : value}. Τα δεδομένα αυτά δεν χρειάζεται να βρίσκονται σε συγκεκριμένη σειρά, μπορούν να γίνουν αλλαγές (όπως προσθήκη ή διαγραφή δεδομένων) αλλά δεν μπορεί να υπάρξει δύο φορές το ίδιο key στην ίδια συλλογή. Όπως αναφέρεται χαρακτηριστικά στη γλώσσα της Python τα dictionaries είναι unordered, changeable και δεν επιτρέπουν duplicates.

LIST (ΛΙΣΤΑ):

Είναι μία συλλογή [25] που αποθηκεύει πολλά στοιχεία δεδομένων (items) σε μία μεταβλητή. Κατ'αναλογία με την παραπάνω ορολογία είναι μία συλλογή ordered, changeable που επιτρέπει duplicates. Δηλαδή, θεωρείται ordered γιατί τα στοιχεία ακολουθούν μία συγκεκριμένη σειρά όπου το πρώτο στοιχείο ξεκινάει από τη θέση [0] , το επόμενο βρίσκεται στη θέση [1] κ.ο.κ. η οποία δεν αλλάζει. Changeable είναι υπό την έννοια ότι τα στοιχεία μπορούν να αλλάζουν, να αυξάνονται ή να αφαιρούνται από αυτήν. Τέλος επιτρέπει να επαναλαμβάνεται η τιμή οποιουδήποτε στοιχείου. Με τον όρο τιμή αναφερόμαστε σε οποιοδήποτε τύπο (type) πάρουν τα δεδομένα της λίστας (αριθμό, χαρακτήρα, τιμή αληθείας).

Ο αλγόριθμος στηρίζεται σε μία πολύ σημαντική αρχή που ονομάζεται ***"Edge Relaxation"*** {[8], [26]}***.***

Πιο συγκεκριμένα,

Έστω ότι εξετάζεται η μετάβαση από έναν κόμβο A σε έναν κόμβο B με κόστος w(A,B).

Αν ισχύει η σχέση:	d[B] > d[A] + w(A, B)
τότε ορίζεται :	d[B] = d[A] + w(A, B) ως το νέο κόστος του κόμβου B.

Αυτό σημαίνει ότι το μονοπάτι μέσω του κόμβου A μπορεί να είναι μικρότερο, καθώς φαίνεται ότι το συνολικό κόστος της διαδρομής μπορεί να είναι χαμηλότερο μέσω του κόμβου A σε σχέση με την τρέχουσα διαδρομή. Ουσιαστικά αυτή η σχέση επαναλαμβάνεται σε κάθε βήμα του αλγορίθμου και καταλήγει στην εύρεση της ελάχιστης διαδρομής όπως θα φανεί παρακάτω.

4.5.2. ΠΕΡΙΓΡΑΦΗ ΚΩΔΙΚΑ

1. Αρχικοποιούμε την απόσταση από τον αρχικό κόμβο πηγή προς όλους τους κόμβους να είναι άπειρη και η απόσταση από τον εαυτό της να είναι ίση με το 0
2. Δημιουργούμε μία κενή λίστα path (διαδρομή)
3. Εισάγουμε ένα ζεύγος (κόμβος, απόσταση) στο Dictionary
4. Όσο το dictionary δεν είναι άδειο τότε:
5. Διαλέγουμε το ζεύγος (κόμβος, απόσταση) που δίνει τη μικρότερη απόσταση από τον τρέχοντα κόμβο και το αφαιρούμε από το Dictionary. Αυτό το ζεύγος στην επόμενη επανάληψη θα θεωρείται ο νέος τρέχων κόμβος.
6. Για κάθε γειτονικό κόμβο προς τον τρέχοντα κόμβο ελέγχουμε τη συνθήκη του “edge relaxation”

Αν,

Απόσταση[γειτονικού κόμβου] > μήκος γειτονικού κόμβου από τον τρέχοντα κόμβο + απόσταση[τρέχοντα κόμβου από αρχικό κόμβο πηγή]

Τότε ανανεώνουμε

Απόσταση[γειτονικού κόμβου] **=** μήκος γειτονικού κόμβου από τον τρέχοντα κόμβο + απόσταση[τρέχοντα κόμβου]

7. Αποθηκεύουμε στη λίστα διαδρομής με τη σειρά τους κόμβους που διασχίσαμε μέχρι να βρεθούμε στον εκάστοτε τελικό κόμβο, έχοντας κάνει την ελάχιστη διαδρομή.

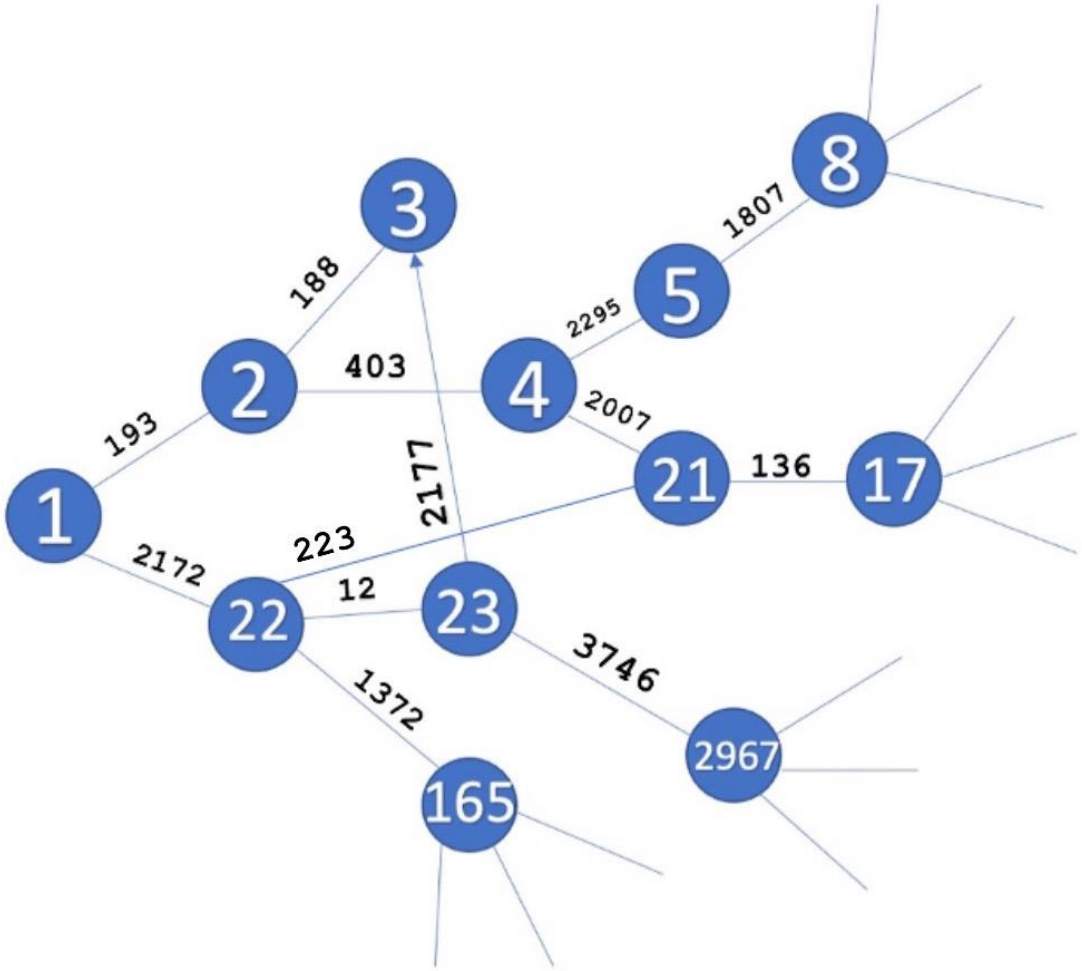
8. Ανανεώνουμε το Dictionary με τα νέα ζευγάρια (γειτονικός κόμβος, νέα απόσταση γειτονικού κόμβου)

Όταν το Dictionary είναι πλέον άδειο ο αλγόριθμος θα σταματήσει και θα έχουμε όλες τις αποστάσεις από κάθε κόμβο προς κάθε άλλο κόμβο του δικτύου.

Ο χρόνος πολυπλοκότητας του αλγορίθμου είναι O$((E+V)\cdot Log(V))$ όπου V είναι ο αριθμός των κόμβων και E ο αριθμός των ακμών.

4.5.3. ΛΥΣΗ ΠΑΡΑΔΕΙΓΜΑΤΟΣ

Στο Παράρτημα Α βρίσκονται τα συνολικά δεδομένα και ο κώδικας. Σε αυτή την ενότητα θα δείξουμε με ποιον τρόπο αναπτύσσεται το δίκτυο αναλύοντας τους πρώτους κόμβους του και θα παρουσιάσουμε τα τελικά αποτελέσματα.



Σχήμα 4.5.1.: Γραφική αναπαράσταση των πρώτων κόμβων του οδικού δικτύου της Ρώμης

ΒΗΜΑ: 1

Αρχικοποιούμε όλες τις αποστάσεις από τον κόμβο 1 να είναι ίσες με άπειρο και η απόσταση από τον εαυτό του να είναι 0.
Εισάγουμε στο Dictionary το πρώτο ζεύγος (κόμβος, απόσταση) να είναι το (1,0).

ΑΡΧΙΚΟΣ ΚΟΜΒΟΣ ΠΗΓΗ	ΤΡΕΧΩΝ ΤΕΛΙΚΟΣ ΚΟΜΒΟΣ	ΑΠΟΣΤΑΣΗ ΑΠΟ ΚΟΜΒΟ ΠΗΓΗ	DICTIONARY	
			κόμβος	απόσταση
1	1	0	1	0
1	2	∞		
1	3	∞		
1	4	∞		
…	…	…		
1	3353	∞		

Πίνακας 4.5.1: Υπολογισμός αποστάσεων πρώτου βήματος και δημιουργία αρχικού Dictionary

ΒΗΜΑ 2:

Αφαιρούμε από το Dictionary το ζεύγος (1,0). Τρέχων κόμβος: 1
Εφαρμόζουμε το edge relaxation στους 2 γειτονικούς κόμβους (adjacent_node) 2 & 22.

Έως τώρα Απόσταση[2] = ∞ & Απόσταση[22]= ∞

Απόσταση[2] > απόσταση[1 → 2] + απόσταση[1] ⟹ ∞ > 193 + 0

Άρα, απόσταση[2] = 139

Αντίστοιχα,

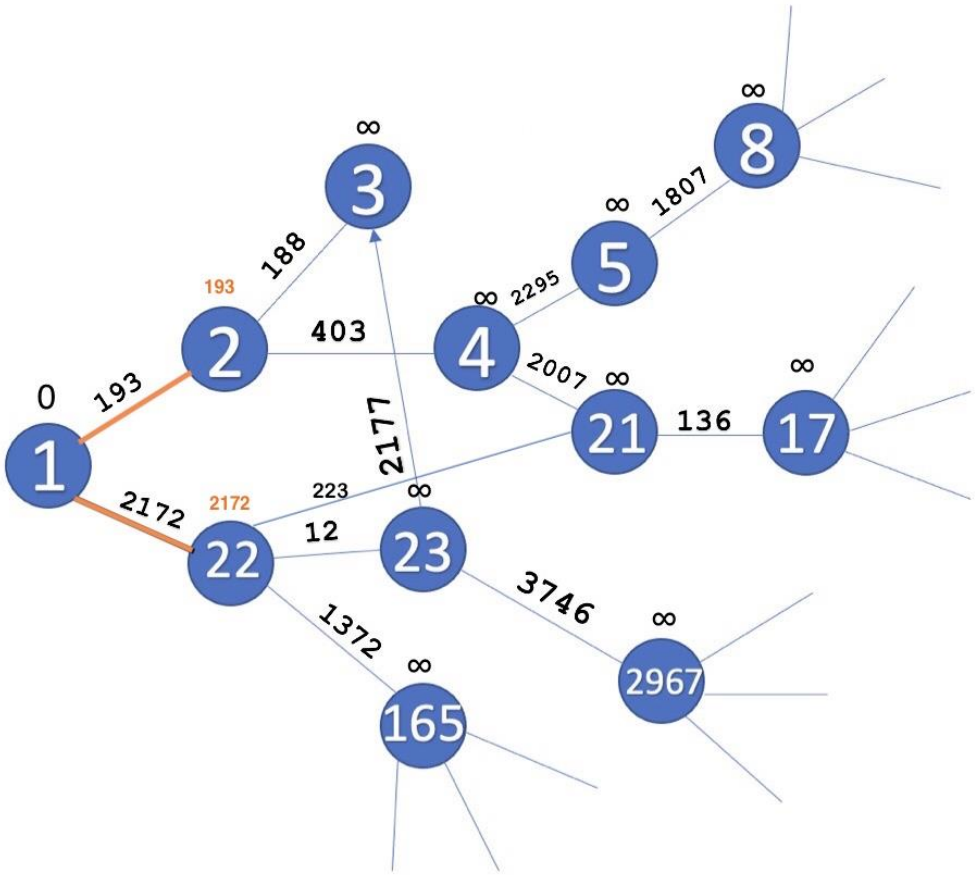
απόσταση[22] > απόσταση[1 → 22] + απόσταση[1] ⟹ ∞ > 2172 + 0

Άρα, απόσταση[22] = 2172

Οι υπολογισμοί και η εισαγωγή νέων ζευγών στο dictionary φαίνονται στον παρακάτω πίνακα:

ΤΡΕΧΩΝ ΚΟΜΒΟΣ	ΓΕΙΤΟΝΙΚΟΣ ΚΟΜΒΟΣ	ΑΠΟΣΤΑΣΤΗ ΓΕΙΤΟΝΙΚΟΥ ΚΟΜΒΟΥ ΑΠΌ ΤΟΝ ΑΡΧΙΚΟ ΚΟΜΒΟ-ΠΗΓΗ	EDGE RELAXATION απόσταση[γειτονικού_κόμβου]> απόσταση[1--> γειτ. κόμβο] + απόσταση[1]]	ΝΕΑ ΑΠΟΣΤΑΣΗ ΓΕΙΤΟΝΙΚΟΥ ΚΟΜΒΟΥ	DICTIONARY	
					κόμβος	απόσταση
1	2	∞	∞ > 193 + 0	193	2	193
	22	∞	∞ > 2172 + 0	2172	22	2172

Πίνακας 4.5.2: Υπολογισμοί 2^{ου} βήματος



Σχήμα 4.5.2: Κατάσταση δικτύου μετά τους υπολογισμούς κόστους διαδρομής του κόμβου 1

ΒΗΜΑ 3:

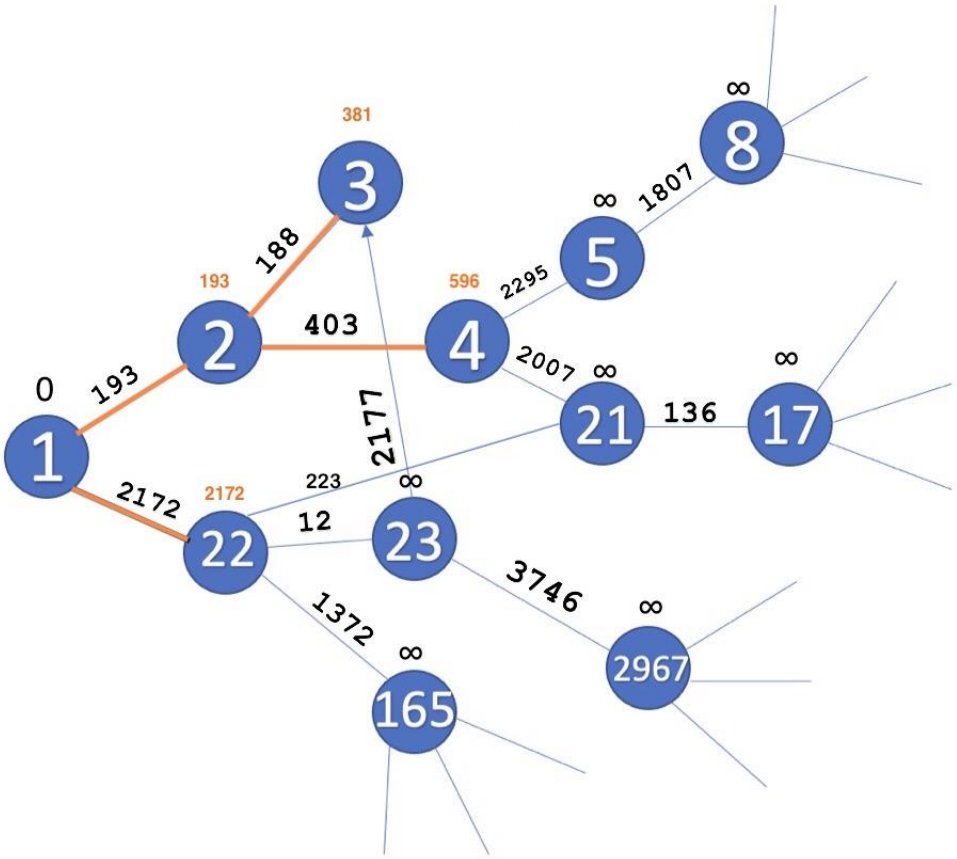
Αφαιρούμε από το Dictionary το ζεύγος (2,193) που έχει το ελάχιστο κόστος. Επομένως, σε αυτό το βήμα ο τρέχων κόμβος είναι το 2. Ακολουθώντας τα παραπάνω βήματα έχουμε τα εξής αποτελέσματα:

ΤΡΕΧΩΝ ΚΟΜΒΟΣ	ΓΕΙΤΟΝΙΚΟΣ ΚΟΜΒΟΣ	ΑΠΟΣΤΑΣΤΗ ΓΕΙΤΟΝΙΚΟΥ ΚΟΜΒΟΥ ΑΠΌ ΤΟΝ ΑΡΧΙΚΟ ΚΟΜΒΟ-ΠΗΓΗ	EDGE RELAXATION απόσταση[γειτονικού_κόμβου]> απόσταση[2--> γειτ. κόμβο] + απόσταση[2]	ΝΕΑ ΑΠΟΣΤΑΣΗ ΓΕΙΤΟΝΙΚΟΥ ΚΟΜΒΟΥ
2	1	0	0 > 193 + 193	0
	3	∞	∞ > 188+193	381
	4	∞	∞ > 403+193	596

Πίνακας 4.5.3: Υπολογισμοί 3^{ου} βήματος

Το ανανεωμένο Dictionary είναι:

DICTIONARY	
κόμβος	απόσταση
22	2172
3	381
4	596



Σχήμα 4.5.3: Κατάσταση δικτύου μετά τους υπολογισμούς κόστους διαδρομής του κόμβου 2

ΒΗΜΑ 4:

Αφαιρούμε τον ζεύγος (3, 381) άρα τρέχων κόμβος : 3

Βλέπουμε ότι ο κόμβος 3 δεν συνδέεται άμεσα με κανέναν άλλον κόμβο εκτός από αυτόν που συνδέθηκε στο προηγούμενο βήμα. Είναι προφανές ότι δεν ικανοποιείται η συνθήκη αφού

απόσταση[2] > απόσταση[3 → 2] + απόσταση[3] ⟹ 193 > 188 + 381= 569 δεν ισχύει.

Άρα απόσταση[2] = 193 (παραμένει ίδια) και η μόνη αλλαγή είναι το ανανεωμένο Dictionary:

DICTIONARY	
κόμβος	απόσταση
22	2172
4	596

ΒΗΜΑ 5:

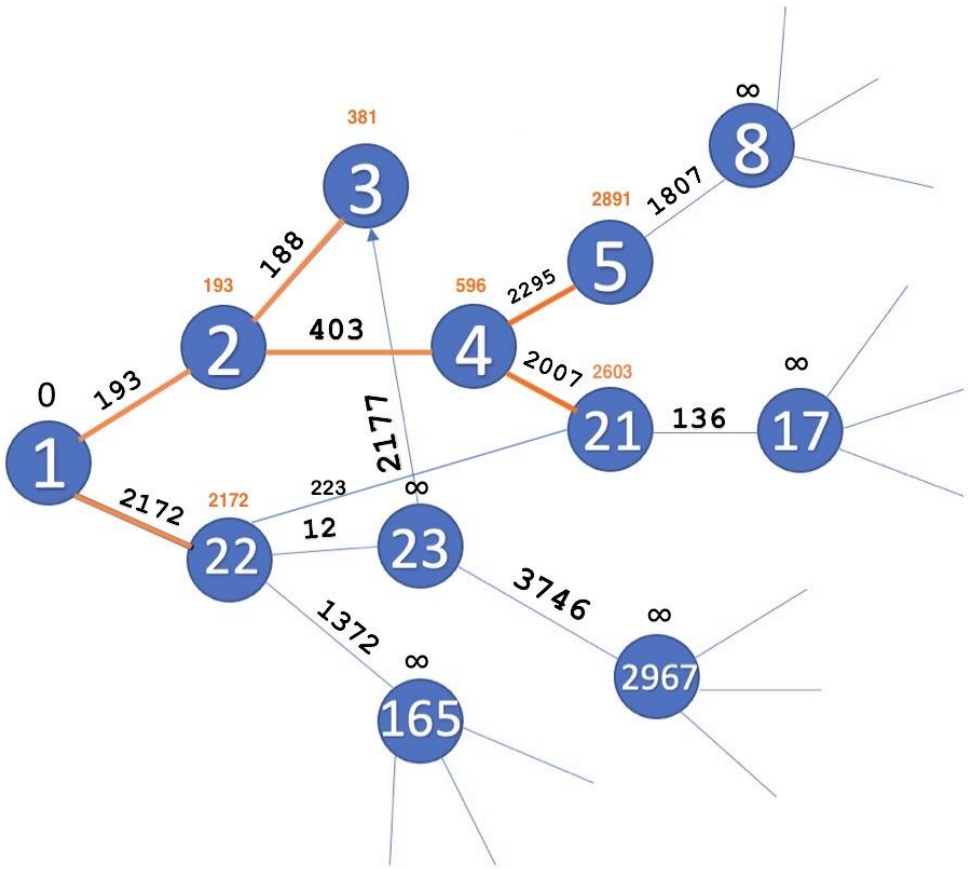
Αφαιρούμε το ζεύγος (4, 596) το οποίο έχει το μικρότερο κόστος. Τρέχων κόμβος είναι πλέον το 4 και έχουμε τα εξής αποτελέσματα:

ΤΡΕΧΩΝ ΚΟΜΒΟΣ	ΓΕΙΤΟΝΙΚΟΣ ΚΟΜΒΟΣ	ΑΠΟΣΤΑΣΤΗ ΓΕΙΤΟΝΙΚΟΥ ΚΟΜΒΟΥ ΑΠΌ ΤΟΝ ΑΡΧΙΚΟ ΚΟΜΒΟ-ΠΗΓΗ	EDGE RELAXATION απόσταση[γειτονικου_κόμβου]> απόσταση[4--> γειτ. κόμβο] + απόσταση[4]	ΝΕΑ ΑΠΟΣΤΑΣΗ ΓΕΙΤΟΝΙΚΟΥ ΚΟΜΒΟΥ
4	5	∞	∞ > 2295 + 596	2891
	21	∞	∞ > 2007 + 596	2603

Πίνακας 4.5.4: Υπολογισμοί 5ου βήματος

Και το Dictionary περιέχει τα παρακάτω ζεύγη:

DICTIONARY	
κόμβος	απόσταση
5	2891
21	2603
22	2172



Σχήμα 4.5.4: Κατάσταση δικτύου μετά τους υπολογισμούς κόστους διαδρομής του κόμβου 4

ΒΗΜΑ 6:

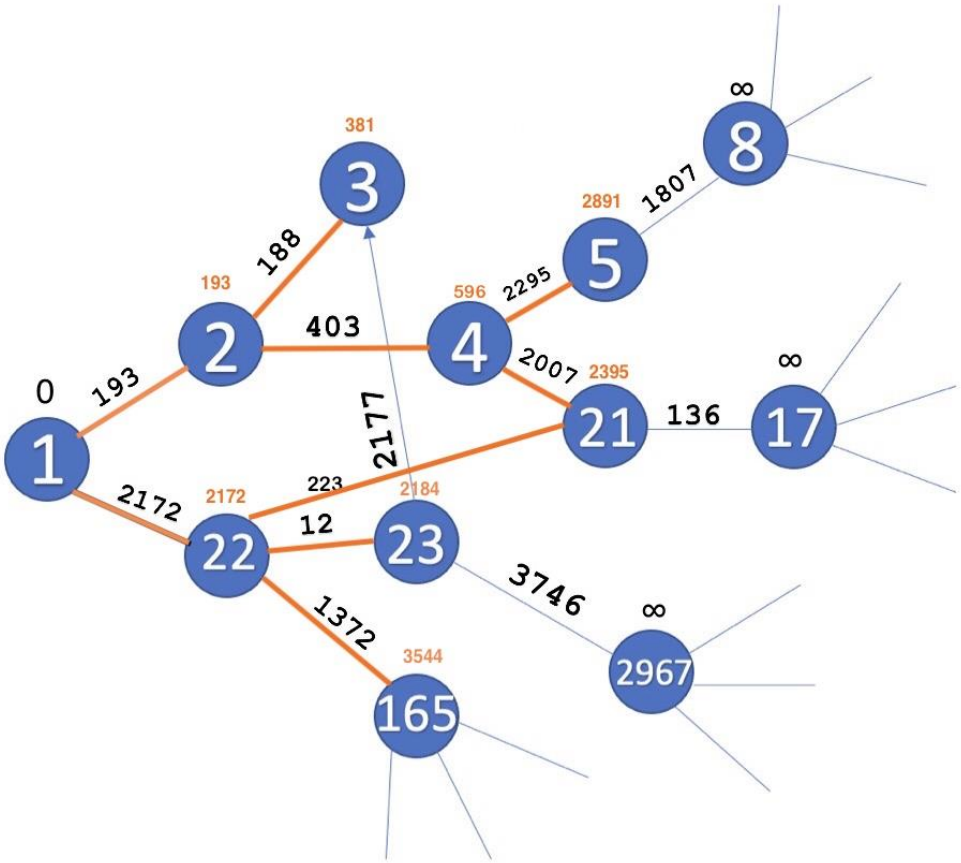
Αφαιρούμε το ζεύγος (22, 2172) και θεωρούμε τον 22 ως τρέχοντα κόμβο.

ΤΡΕΧΩΝ ΚΟΜΒΟΣ	ΓΕΙΤΟΝΙΚΟΣ ΚΟΜΒΟΣ	ΑΠΟΣΤΑΣΤΗ ΓΕΙΤΟΝΙΚΟΥ ΚΟΜΒΟΥ ΑΠΌ ΤΟΝ ΑΡΧΙΚΟ ΚΟΜΒΟ-ΠΗΓΗ	EDGE RELAXATION απόσταση[γειτονικού_κόμβου]> απόσταση[22--> γειτ. κόμβο] + απόσταση[22]	ΝΕΑ ΑΠΟΣΤΑΣΗ ΓΕΙΤΟΝΙΚΟΥ ΚΟΜΒΟΥ
22	21	2603	2603 > 223 + 2172	2395
	23	∞	∞ > 12 + 2172	2184
	165	∞	∞ > 1372 + 2172	3544

Πίνακας 4.5.5: Υπολογισμοί 6ου βήματος

Με το ανανεωμένο Dictionary:

DICTIONARY	
κόμβος	απόσταση
5	2891
21	2395
23	2184
165	3544



Σχήμα 4.5.5: Κατάσταση δικτύου μετά τους υπολογισμούς κόστους διαδρομής του κόμβου 22

Στον παρακάτω πίνακα[7] παρουσιάζεται η ελάχιστη διαδρομή που ακολουθείται από τον αρχικό κόμβο-πηγή (1) προς κάθε έναν κόμβο ξεχωριστά, ο οποίος θεωρείται σε κάθε βήμα τελικός κόμβος. Έχουμε λοιπόν:

ΑΠΌ ΑΡΧΙΚΟ ΚΌΜΒΟ ΠΗΓΗ	ΔΙΑΔΡΟΜΗ -->	ΣΤΟΝ ΤΕΛΙΚΟ ΚΟΜΒΟ
1	-	1
1	2	2
1	2	3
1	2	4
1	2--> 4	5
…	…	…
1	-	22
1	22	23
…	…	…
1	22	165

Πίνακας 4.5.6: Υπολογισμοί ελάχιστης διαδρομής από τον κόμβο 1 προς τους υπόλοιπους κόμβους

Από τον πίνακα 4.5.6 συμπεραίνουμε ότι ξεκινώντας από τον αρχικό κόμβο-πηγή φτάνουμε απευθείας στον κόμβο 2. Για να φτάσουμε στους κόμβους 3 και 4, αντίστοιχα (πάλι ξεκινώντας από τον κόμβο-πηγή) περνάμε από τον κόμβο 2. Για να φτάσουμε στον κόμβο 5 περνάμε διαδοχικά από τους κόμβους 2 και 4. Ανάλογα, βρίσκουμε τις υπόλοιπες ελάχιστες διαδρομές πάντα ξεκινώντας από τον αρχικό κόμβο-πηγή προς κάθε άλλο κόμβο του δικτύου.

Η ελάχιστη διαδρομή όπως αυτή βγήκε από τον κώδικα ξεκινώντας από τον αρχικό κόμβο 1 μέχρι τον τελικό κόμβο 3353 είναι:

1→22→165→162→167→164→171→190→191→336→338→343→344→340→347→348→335→515→407→524→582→589→596→597→636→641→642→632→649→655→1229→1442→1277→1473→1475→2392→3353

Με ελάχιστο κόστος: 30305

[7] Παρουσιάζονται τα αποτελέσματα μόνο για τους κόμβους που λύσαμε παραπάνω, όχι για τους συνολικούς κόμβους του προβλήματος.

ΚΕΦΑΛΑΙΟ 5

5.1. ΠΡΟΒΛΗΜΑ ΠΛΑΝΟΔΙΟΥ ΠΩΛΗΤΗ / TRAVELLING SALESMAN PROBLEM

Το πρόβλημα του πλανόδιου πωλητή (Travelling Salesman Problem- TSP) είναι ένα από τα πιο μελετημένα προβλήματα συνδυαστικής βελτιστοποίησης (combinatorial optimization) στα πεδία της επιχειρησιακής έρευνας και της επιστήμης υπολογιστών {[27], [30]}. Η περιγραφή του προβλήματος συνοψίζεται ως εξής: δοσμένου ενός αριθμού n πόλεων και των αποστάσεων μεταξύ κάθε ζεύγους πόλεων ποια είναι η ελάχιστη διαδρομή που μπορεί να ακολουθήσει ο πλανόδιος πωλητής προκείμενου να περάσει από όλες αυτές τις πόλεις ακριβώς μία φορά επιστρέφοντας στην πόλη από την οποία ξεκίνησε. Χαρακτηρίζεται ως ένα πρόβλημα κλάσης NP-Hard και όχι NP-Complete όπως συχνά παρουσιάζεται [28]. Αυτός ο διαχωρισμός γίνεται καθώς για να είναι ένα πρόβλημα NP-Complete πρέπει να ανήκει και στην κλάση NP αλλά να είναι και NP-Hard, όπως αναφέραμε στην ενότητα 2.4. Εδώ, δεν παρουσιάζεται ένα πρόβλημα απόφασης, αλλά βελτιστοποίησης, το οποίο δεν είναι κλάσης NP καθώς ούτε η λύση αλλά ούτε η επαλήθευση των λύσεων μπορεί να γίνει σε πολυωνυμικό χρόνο. Πιο συγκεκριμένα, προκειμένου να επαληθεύσουμε τις λύσεις δεν μπορούμε να αποφύγουμε την εύρεση της συνολικής βέλτιστης λύσης, η οποία όπως ξέρουμε δεν μπορεί να βρεθεί σε πολυωνυμικό χρόνο. Αντίθετα, υπάρχει μία παραλλαγή του TSP, ένα πρόβλημα απόφασης, το οποίο είναι NP-Complete [29]. Στόχος αυτού είναι ο έλεγχος ύπαρξης μίας διαδρομής της οποίας το συνολικό μήκος να είναι μικρότερο από ένα δοσμένο αριθμό. Από τις αρχές του 1950 το πρόβλημα του πλανόδιου πωλητή άρχισε να μελετάται εντατικά αλλά μέχρι σήμερα δεν έχει βρεθεί κάποιος αλγόριθμος που να είναι πλήρως αποτελεσματικός όσο αυξάνεται η πολυπλοκότητα του προβλήματος, όσο δηλαδή αυξάνουμε τους προορισμούς [31]. Ωστόσο, έχει βρεθεί ένας μεγάλος αριθμός τεχνικών και μεθόδων που επιλύουν το πρόβλημα. Μπορούν να βρεθούν είτε ακριβείς λύσεις μέσω ακριβών αλγορίθμων (exact algorithms) είτε προσεγγιστικές λύσεις μέσω προσεγγιστικών αλγορίθμων (approximate and heuristic[8] algorithms). Παρακάτω θα παρουσιάσουμε την επίλυση του προβλήματος μέσω ενός αλγορίθμου δυναμικού προγραμματισμού που ονομάζεται “Held and Karp algorithm”.

5.2. ΠΕΡΙΓΡΑΦΗ ΠΡΟΒΛΗΜΑΤΟΣ TSP

Το πρόβλημα του πλανόδιου πωλητή παρουσιάζεται [27] από ένα πλήρες[9] γράφημα G(V, E) όπου V είναι το σύνολο των κόμβων (έστω |V|=n), το οποίο αναπαριστά τις πόλεις, τις περιοχές ή οποιαδήποτε σημεία πρόκειται να επισκεφθεί ο πλανόδιος πωλητής και E το σύνολο των ακμών που δηλώνουν τη διαδρομή για τη μετάβαση από τον έναν κόμβο στον άλλον. Με C=c(i, j) συμβολίζουμε το κόστος (απόσταση στη συγκεκριμένη περίπτωση) για τη μετάβαση από έναν κόμβο i σε έναν κόμβο j το οποίο είναι μη αρνητικό. Το TSP μπορεί να είναι είτε συμμετρικό είτε μη συμμετρικό αναλόγως τον τύπο του γραφήματος. Στην πρώτη

[8] Προσεγγιστικοί αλγόριθμοι για την εύρεση προσεγγιστικής λύσης όταν οι κλασικές μέθοδοι αδυνατούν να βρουν την ακριβή λύση. Θυσιάζουν ακρίβεια στη λύση προκειμένου να επιλυθούν γρηγορότερα [65].

[9] Πλήρες ονομάζεται το γράφημα n κορυφών όπου όλα τα ζεύγη κορυφών συνδέονται με μία ακμή, δηλαδή υπάρχουν συνολικά n(n-1)/2 ακμές [17].

περίπτωση η απόσταση μεταξύ δύο κόμβων A & B είναι ίση, ανεξαρτήτως κατεύθυνσης, δηλαδή είτε γίνει μετάβαση από το A στο B είτε το αντίστροφο. Έτσι, σχηματίζεται ένα μη κατευθυνόμενο, διπλής κατεύθυνσης (bidirectional) γράφημα. Στην περίπτωση του μη συμμετρικού TSP παρουσιάζεται ένα κατευθυνόμενο γράφημα στο οποίο μπορεί είτε η απόσταση μεταξύ δύο κόμβων να είναι διαφορετική αναλόγως την κατεύθυνση μετάβασης, είτε να υπάρχει διαδρομή μόνο προς τη μία κατεύθυνση. Σε πραγματικά προβλήματα αυτή η ασυμμετρία μπορεί να οφείλεται στην ύπαρξη μονόδρομων ή ακόμα και στα διαφορετικά κόστη αεροπορικών εισιτηρίων για πόλεις με διαφορετικά τέλη αναχώρησης ή άφιξης [32].

5.3. ΠΡΟΒΛΗΜΑ ΠΛΑΝΟΔΙΟΥ ΠΩΛΗΤΗ ΜΕ ΜΕΘΟΔΟΥΣ ΔΥΝΑΜΙΚΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Ο δυναμικός προγραμματισμός στο πρόβλημα του πλανόδιου πωλητή είναι μία αποτελεσματική μέθοδος για την εύρεση μίας ακριβούς λύσης. Οι Held και Karp, το 1962 ανέπτυξαν έναν αλγόριθμο [33], τον λεγόμενο «Held and Karp algorithm» ο οποίος μειώνει κατά πολύ την πολυπλοκότητα, δηλαδή το χρόνο και το χώρο εύρεσης μίας τέτοιας λύσης. Αρκεί να σκεφτούμε ότι μία άλλη μέθοδος εύρεσης ακριβούς λύσης όπως «η προς τα εμπρός μέθοδος» (brute force method) χρειάζεται (n-1)! μεταθέσεις προκειμένου να καλύψει όλους τους n κόμβους και να ελέγξει όλες τις πιθανές διαδρομές μέχρι να καταλήξει στην ελάχιστη. Δηλαδή έχει χρόνο εκτέλεσης του αλγορίθμου O(n!). Αυτό πρακτικά σημαίνει ότι είναι αδύνατο να εφαρμόσουμε μία τέτοια μέθοδο ακόμα και για έναν σχετικά μικρό αριθμό κόμβων. Για παράδειγμα αν έχουμε ένα πρόβλημα 4 κόμβων θα χρειαστούν (3)! = 6 μεταθέσεις (1->2, 1->3, 1->4, 2->3, 2->4, 3->4) ενώ για μόλις 10 κόμβους υπάρχουν 9! = 362,880 πιθανοί συνδυασμοί διαδρομών πριν φτάσουμε στον αρχικό κόμβο. Λέμε λοιπόν ότι ο χρόνος πολυπλοκότητας (time complexity) μίας τέτοιας προσέγγισης είναι O(n!) και ο χώρος (space complexity) $O(n^2)$ {[33], [34]}. Αντίθετα, με τη μέθοδο του δυναμικού προγραμματισμού ο χρόνος πολυπλοκότητας είναι $O(2^n n^2)$ και ο χώρος $O(2^n n)$, όπου n ο αριθμός των κόμβων {[33], [34]}. Το πλεονέκτημα λοιπόν, είναι ότι ο αλγόριθμος H&K προσφέρει λύση γρηγορότερα από τη μέθοδο “brute force”, όμως καθώς αυξάνεται ο αριθμός των κόμβων, αυξάνεται και ο χρόνος εκθετικά και γι’ αυτό είναι αποτελεσματικός σε προβλήματα που δεν ξεπερνούν τους 20 κόμβους. Σαφώς υπάρχουν μέθοδοι (ακριβείς και προσεγγιστικοί) οι οποίοι επιλύουν αποτελεσματικά το πρόβλημα για μεγαλύτερο αριθμό κόμβων [35]. Παρολ΄ αυτά στην παρούσα μελέτη θα περιοριστούμε στη επίλυση του TSP μέσω του δυναμικού προγραμματισμού εφαρμόζοντας τον αλγόριθμο Held & Karp.

5.3.1. ΑΛΓΟΡΙΘΜΟΣ HELD AND KARP

Για απλούστευση της ερμηνείας του προβλήματος θα θεωρούμε πλέον τους κόμβους να είναι πόλεις και το κόστος για τη μετάβαση από έναν κόμβο i σε έναν κόμβο j να δηλώνει τη μεταξύ τους απόσταση. Η κεντρική ιδέα είναι ο διαχωρισμός του προβλήματος σε μικρότερα υπό-προβλήματα έτσι ώστε κάθε επόμενο βήμα να περιέχει τα προηγούμενα, ήδη υπολογισμένα, βήματα. Με αυτόν τον τρόπο αποφεύγεται η επανάληψη του υπολογισμού κάποιων βημάτων που θα οδηγούσαν σε ακόμα πιο αυξημένο χρόνο πολυπλοκότητας.

Ο αλγόριθμος βασίζεται στην παρακάτω αρχή [66]:

Κάθε μικρότερη διαδρομή που περιέχεται στην κανονική διαδρομή ελάχιστου κόστους, είναι από μόνη της ελάχιστου κόστους

Έστω V={1, 2, ..., n} το σύνολο των πόλεων και n ο αριθμός τους. Χωρίς βλάβη της γενικότητας ορίζεται η πόλη 1 ως αρχικός και τελικός προορισμός.
Ορίζονται τα εξής {[33], [36]}:

S $\subset V\setminus\{1\} \equiv (2, \dots, n)$: ένα υποσύνολο μεγέθους s ($1 \leq s \leq n-1$) το οποίο περιλαμβάνει τις πόλεις από τις οποίες πρέπει να περάσουμε ακριβώς μία φορά όταν βρισκόμαστε σε κάποια πόλη i.

d(i, j) : η απόσταση της πόλης i από την πόλη j

f(S, i) : το κόστος/μήκος της ελάχιστης διαδρομής από την πόλη 1 μέχρι την πόλη i περνώντας ακριβώς μία φορά από τις πόλεις του υποσυνόλου S.

Βρίσκουμε το κόστος αυτό για κάθε υπό-πρόβλημα, δηλαδή για κάθε πιθανή πόλη i μέχρι να έχουμε επισκεφτεί κάθε πόλη ακριβώς μία φορά (εκτός από την αρχική πόλη 1).

Η αναδρομική σχέση του προβλήματος ορίζεται ως εξής:

$$f(S,i) = \begin{cases} \min\limits_{j\in S-\{i\}} \{\, (f(S\setminus\{i\}, j) + d(j,i)\}, & \alpha\nu\ |S| > 1 \\ d(1,i), & \alpha\nu\ |S| = 1 \end{cases} \qquad (5.1)$$

Στο τελικό στάδιο όπου πλέον έχουμε επισκεφτεί όλες τις πόλεις μία φορά και μένει να επιστρέψουμε στην αρχική πόλη το τελικό ελάχιστο κόστος διαδρομής παρουσιάζεται από την εξής σχέση:

$$F(S,i) = \min_{j\in S}\{\, (f(S)\,, i) + d(i,1) \text{ όπου S=(2,..,n)} \qquad (5.2)$$

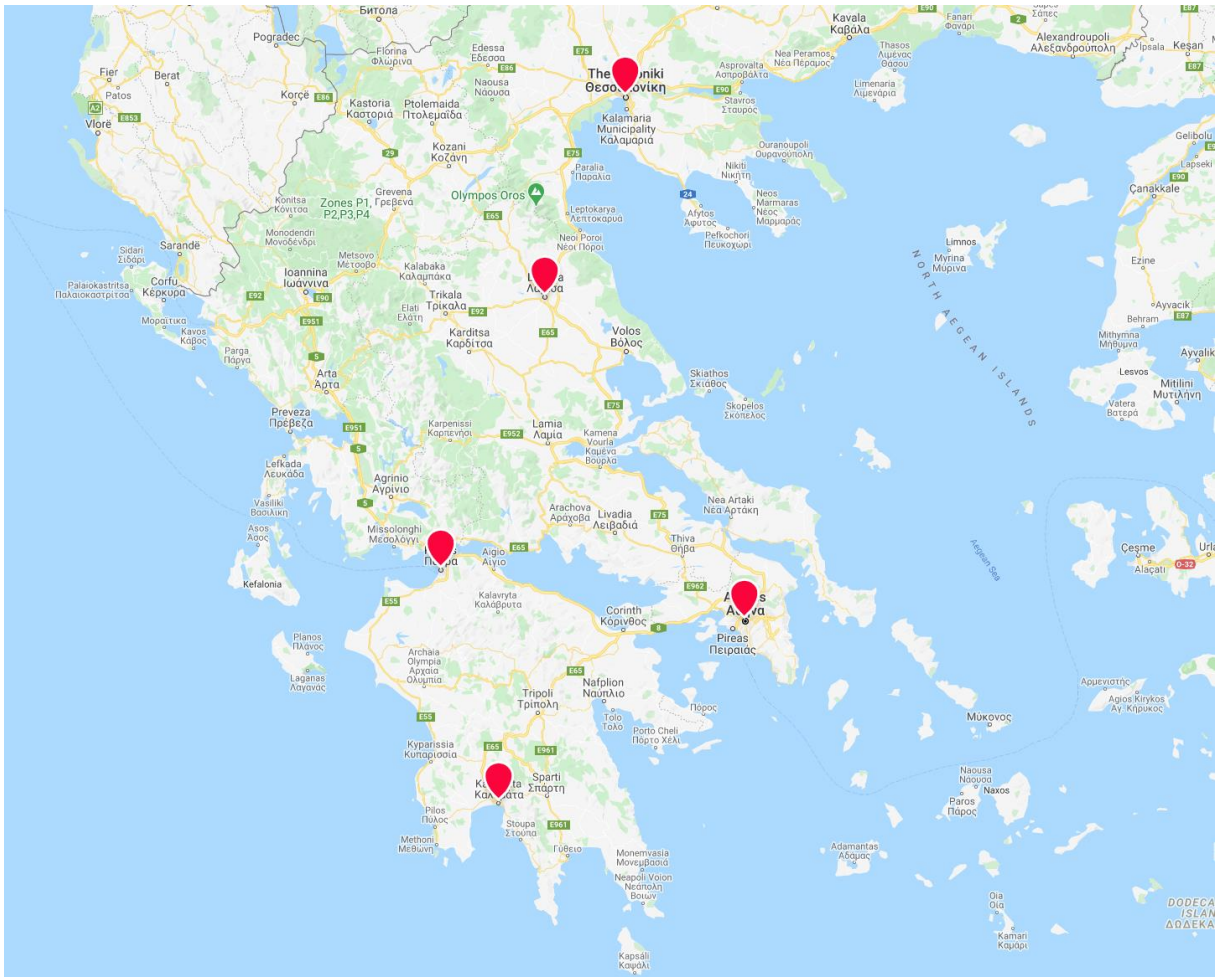
5.4. ΠΡΟΒΛΗΜΑ

Μια εταιρεία ρούχων στην Αθήνα έχει δεχτεί παραγγελίες από πελάτες που βρίσκονται σε 5 διαφορετικές πόλεις της Ελλάδας. Στόχος της είναι να μεταφέρει τις παραγγελίες στον προορισμό τους ακολουθώντας την ελάχιστη διαδρομή ξεκινώντας από την Αθήνα επιστρέφοντας πάλι πίσω. Στον παρακάτω πίνακα φαίνονται οι αποστάσεις (σε χιλιόμετρα) από κάθε πόλη i προς κάθε πόλη j.

		ΑΠΟΣΤΑΣΗ (ΧΙΛΙΟΜΕΤΡΑ)				
		1	2	3	4	5
ΠΟΛΕΙΣ		ΑΘΗΝΑ	ΠΑΤΡΑ	ΘΕΣΣΑΛΟΝΙΚΗ	ΛΑΡΙΣΑ	ΚΑΛΑΜΑΤΑ
1	ΑΘΗΝΑ	0	211	502	345	239
2	ΠΑΤΡΑ	211	0	462	322	211
3	ΘΕΣΣΑΛΟΝΙΚΗ	502	462	0	152	728
4	ΛΑΡΙΣΑ	345	322	152	0	581
5	ΚΑΛΑΜΑΤΑ	239	211	728	581	0

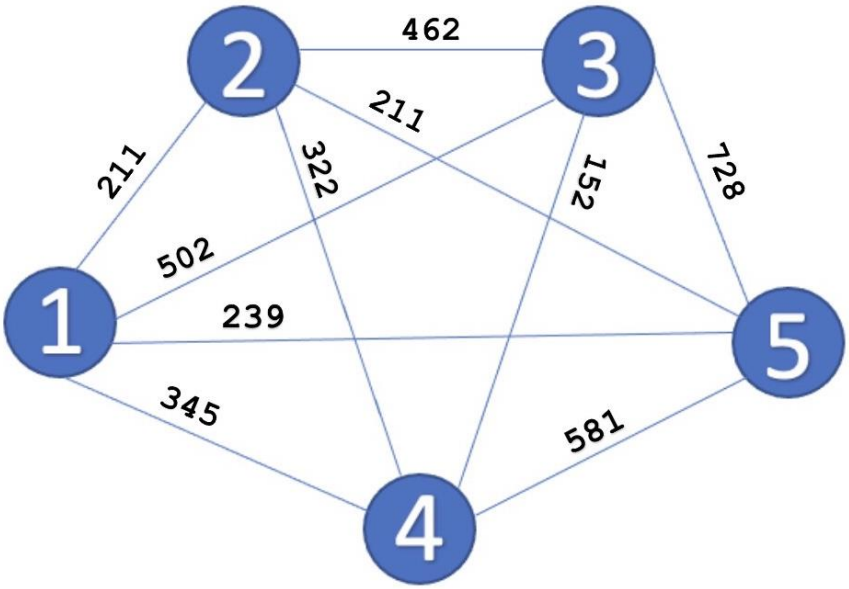
Πίνακας 5.4.1: Αποστάσεις (σε χιλιόμετρα) [37] από κάθε μία πόλη προς κάθε άλλη

Οι πόλεις φαίνονται στον παρακάτω χάρτη



Εικόνα 5.4.1: Χάρτης Ελλάδας όπου απεικονίζονται οι 5 πόλεις προς επίσκεψη [38]

Το πρόβλημα είναι ένα συμμετρικό TSP που αποτελείται από 5 κόμβους (πόλεις) που συνδέονται όλοι μεταξύ τους με ακμές πάνω στις οποίες δηλώνονται οι αποστάσεις. Το πλήρες, μη-κατευθυνόμενο γράφημα φαίνεται παρακάτω.



Σχήμα 5.4.1: Γραφική αναπαράσταση των 5 πόλεων (1-Αθήνα, 2-Πάτρα, 3-Θεσσαλονίκη, 4-Λάρισα, 5-Καλαμάτα)

Για την εύρεση της ελάχιστης διαδρομής σημειώνουμε σε κάθε υπολογισμό το εξής:

p(i, S): η επόμενη πόλη κατά σειρά αφού επισκεφτούμε την πόλη i για την οποία επιτυγχάνεται το ελάχιστο κόστος διαδρομής, από τις πόλεις του συνόλου S.

k=0, το S είναι κενό και ισχύει:

f(2, Ø)= d(2, 1)= 211
f(3, Ø)= d(3, 1)= 502
f(4, Ø)= d(4, 1) = 345
f(5, Ø)= d(5, 1)= 239

k=1, θεωρούμε ότι το S περιέχει σύνολα ενός στοιχείου, δηλαδή S= ({2}, {3}, {4} ή {5})

Για S= {2}:

f(3, {2})= d(3, 2) + f(2, Ø)= 462+ 211 = 673 , είναι το μήκος της ελάχιστης διαδρομής ξεκινώντας από την πόλη 1 (Αθήνα) μέχρι την πόλη 3 (Θεσσαλονίκη) περνώντας ακριβώς μία φορά από την πόλη 2 (Πάτρα).

f(4, {2})= d(4, 2) + f(2, Ø)= 322 + 211=533, είναι το μήκος της ελάχιστης διαδρομής ξεκινώντας από την πόλη 1 (Αθήνα), μέχρι την πόλη 4 (Λάρισα) περνώντας ακριβώς μία φορά από την πόλη 2 (Πάτρα).

f(5, {2})= d(5, 2) + f(2, Ø) = 211 + 211= 422, το μήκος της ελάχιστης διαδρομής από την πόλη 1 (Αθήνα), μέχρι την πόλη 5 (Καλαμάτα) περνώντας ακριβώς μία φορά από την πόλη 2 (Πάτρα).

p(3, {2}) = 2, σταματώντας στην πόλη 3, η μικρότερη διαδρομή επιτυγχάνεται αν επισκεφτούμε αμέσως μετά την πόλη 2
p(4, {2}) = 2, σταματώντας στην πόλη 4, η μικρότερη διαδρομή επιτυγχάνεται αν επισκεφτούμε αμέσως μετά την πόλη 2
p(5, {2}) = 2, σταματώντας στην πόλη 5, η μικρότερη διαδρομή επιτυγχάνεται αν επισκεφτούμε αμέσως μετά την πόλη 2

Για S= {3}:

f(2, {3})= d(2, 3) + f(3, Ø)= 462+ 502= 964
f(4, {3})= d(4, 3) + f(3, Ø)= 152+ 502= 654
f(5, {3})= d(5, 3) + f(3, Ø)= 728 + 502= 1230

p(2, {3})= 3
p(4, {3})= 3
p(5, {3})= 3

Για S= {4}:

f(2, {4})= d(2, 4) + f(4, Ø)= 322+ 345= 667
f(3, {4})= d(3, 4) + f(4, Ø)= 152+ 345= 497
f(5, {4})= d(5, 4) + f(4, Ø)= 581+ 345= 926

p(2, {4})= 4
p(3, {4})= 4
p(5, {4})= 4

Για S= {5}:

f(2, {5})= d(2, 5) + f(5, Ø)= 211+ 239= 450
f(3, {5})= d(3, 5) + f(5, Ø)= 728+ 239= 967
f(4, {5})= d(4, 5) + f(5, Ø)= 581+ 239= 820

p(2, {5}) = 5
p(3, {5}) = 5
p(4, {5}) = 5

k=2, θεωρούμε ότι το S περιέχει σύνολα δύο στοιχείων, δηλαδή S= ({2, 3}, {2, 4}, {2, 5}, {3, 4}, {3, 5} ή {4, 5})

Για S={2, 3}:

f(4, {2, 3})= $\min$(d(4, 2) + f(2, {3}), d(4, 3) + f(3, {2})= min(322+ 964, 152+ 673) =
= min(1286, 825) = 825

Είναι το μήκος της ελάχιστης διαδρομής ξεκινώντας από την πόλη 1 (Αθήνα), σταματώντας στην 4 (Λάρισα) περνώντας μία φορά από τις πόλεις 2 & 3.

f(5, {2, 3})= $\min$(d(5, 2) + f(2, {3}), d(5, 3) + f(3, {2})=min(211+ 964, 728+ 673)=
= min(1175, 1401) = 1175

Είναι το μήκος της ελάχιστης διαδρομής ξεκινώντας από την πόλη 1, σταματώντας στην πόλη 5 περνώντας μία φορά από τις πόλεις 2 & 3.

p(4, {2, 3}) = 3, δηλαδή όταν βρισκόμαστε στην πόλη 4, η ελάχιστη διαδρομή προκειμένου να επιστρέψουμε στην πόλη 1, επιτυγχάνεται αν επισκεφτούμε αμέσως μετά την 4, την πόλη 3, δηλαδή προσωρινή ελάχιστη διαδρομή θεωρείται η
4 → 3 → 2 → 1

p(5, {2, 3}) = 3, δηλαδή όταν βρισκόμαστε στην πόλη 5, η ελάχιστη διαδρομή προκειμένου να επιστρέψουμε στην πόλη 1, επιτυγχάνεται αν επισκεφτούμε αμέσως μετά την 5, την πόλη 3, δηλαδή προσωρινή ελάχιστη διαδρομή θεωρείται η 5 → 3 → 2 → 1

Για S={2, 4}:

f(3, {2, 4})= $\min$(d(3, 2) + f(2, {4}), d(3, 4) + f(4, {2})= min(462+ 667, 152+ 533)=
= min(1129,685) = 685

f(5, {2, 4})= $\min$(d(5, 2) + f(2, {4}), d(5, 4) + f(4, {2})= min(211 +667, 581 +533)=
= min(878,1114) = 878

p(3, {2, 4})= 4
p(5, {2, 4})= 2

Για S= {2, 5}:

f(3, {2, 5})= $\min$(d(3, 2) + f(2, {5}), d(3, 5) + f(5, {2})= min(462 + 450, 728 + 422)=
= min(912, 1150) = 912

f(4, {2, 5})= $\min$(d(4, 2) + f(2, {5}), d(4, 5) + f(5, {2})= min(322+ 450, 581+ 422)=
= min(772,1003)= 772

p(3, {2, 5})= 2
p(4, {2, 5})= 2

Για S= {3, 4}:

f(2, {3,4})= $\min$(d(2, 3) + f(3, {4}), d(2, 4) + f(4, {3})= min(462+ 497, 322+ 654)=
= min(959, 976) = 959

f(5, {3, 4})= $\min$(d(5, 3) + f(3, {4}), d(5, 4) + f(4, {3})= min(728 +497, 581+654)=
= min(1225, 1235) = 1225

p(2, {3, 4})= 3
p(5, {3, 4})= 3

Για S= {3, 5}:

f(2, {3, 5})= $\min$(d(2, 3) + f(3, {5}), d(2, 5) + f(5, {3})= min(462+ 967, 211+ 1230) =
= min(1429, 1441) = 1429

f(4, {3, 5})= $\min$(d(4, 3) + f(3, {5}), d(4, 5) + f(5, {3})= min(152+967, 581+ 1230)=
= min(1119, 1811)= 1119

p(2, {3, 5})= 3
p(4, {3, 5})= 3

Για S= {4, 5}:

f(2, {4, 5})= $\min$(d(2, 4) + f(4, {5}), d(2, 5) + f(5, {4})= min(322+ 820, 211+ 926)=
= min(1142, 1137) = 1137

f(3, {4, 5})= $\min$(d(3, 4) + f(4, {5}), d(3, 5) + f(5, {4})= min(152+820, 728+ 926)=
= min(972, 1654)= 972

p(2, {4, 5})= 5
p(3, {4, 5})= 4

k=3, θεωρούμε ότι το S περιέχει σύνολα τριών στοιχείων, δηλαδή S= {2, 3, 4}, {2, 3, 5}, {2, 4, 5} ή {3, 4, 5}

Για S={2, 3, 4}:

f(5, {2, 3, 4})= $\min$(d(5, 2) + f(2, {3, 4}), d(5, 3) + f(3, {2, 4}, d(5, 4) + f(4, {2, 3})=
= min(211+ 959, 728+ 685, 581+ 825) = min(1170, 1413, 1406)= 1170

Είναι το μήκος της ελάχιστης διαδρομής ξεκινώντας από την πόλη 1, σταματώντας στην πόλη 5 περνώντας μία φορά από τις πόλεις 2, 3 & 4.

p(5, {2, 3, 4})= 2

Δηλαδή, όταν βρισκόμαστε στην πόλη 5, η ελάχιστη διαδρομή μέχρι να επιστρέψουμε στην αρχική πόλη 1 επιτυγχάνεται αν επισκεφτούμε αμέσως μετά την 5, την πόλη 2, δηλαδή προσωρινή ελάχιστη διαδρομή θεωρείται η 5 → 2 → {3, 4} → 1
Μετά την πόλη 2 θα μπορούσε να ακολουθεί είτε η 3^{η} είτε η 4^{η} πόλη κάτι το οποίο δεν μας ενδιαφέρει να παρουσιάσουμε σε αυτό το βήμα ωστόσο ξέρουμε ότι έχει ήδη υπολογιστεί.

Για S={2, 3, 5}:

f(4, {2, 3, 5})= $\min$(d(4, 2) + f(2, {3, 5}), d(4, 3) + f(3, {2, 5}, d(4, 5) + f(5, {2, 3})=
=min(322+ 1429, 152+ 912, 581+ 1175) = min(1751, 1064, 1756)= 1064

p(4, {2, 3, 5})= 3

Για S={2, 4, 5}:

f(3, {2, 4, 5})= $\min$(d(3, 2) + f(2, {4, 5}), d(3, 4) + f(4, {2, 5}, d(3, 5) + f(5, {2, 4})=
= min(462+ 1137, 152+ 772, 728+ 878) =min(1599, 924, 1606) = 924

p(3, {2, 4, 5})= 4

Για S={3, 4, 5}:

f(2, {3, 4, 5})= $\min$(d(2, 3) + f(3, {4, 5}), d(2, 4) + f(4, {3, 5}, d(2, 5) + f(5, {3, 4})=
= min(462+ 972, 322+ 1119, 211+ 1225)= min(1434, 1441, 1436)= 1434

p(2, {3, 4, 5})= 3

Μήκος βέλτιστης διαδρομής:

f(1,{2,3,4,5})=min(d(1,2)+f(2,{3,4,5}),d(1,3)+ f(3,{2,4,5}),d(1,4)+ f(4,{2,3,5}),d(1,5)+ f(5,{2,3,4})
=min(211+ 1434, 502+ 924, 345+ 1064, 239+ 1170)=
= min(1645, 1426, 1409, 1409)= 1409

Αυτό είναι το συνολικό κόστος της διαδρομής που ξεκινάει από την Αθήνα περνάει ακριβώς μία φόρα από τις πόλεις (Πάτρα, Θεσσαλονίκη, Λάρισα και Καλαμάτα) και επιστρέφει πίσω στην Αθήνα.

p(1, {2, 3, 4 , 5})= 4 ή 5 (1)

Θεωρώντας ότι έχουμε φτάσει στη βάση ξεκινάμε την εύρεση της βέλτιστης διαδρομής.
Όπως φαίνεται από την σχέση 1, από την πόλη 1 (Αθήνα) έχουμε την επιλογή να επισκεφτούμε πρώτη οποιαδήποτε από τις πόλεις 2, 3, 4 και 5 προτού επιστρέψουμε πάλι πίσω. Το ελάχιστο κόστος εμφανίζεται αν επισκεφτούμε πρώτη είτε την πόλη 4 (Λάρισα) είτε την πόλη 5 (Καλαμάτα).

Περίπτωση 1: Πρώτη μετάβαση $\Longrightarrow$ Λάρισα (4)

Βρισκόμενοι στην πόλη 4 έχουμε τη δυνατότητα να επισκεφτούμε τις πόλεις 2, 3 και 5 (Πάτρα, Θεσσαλονίκη και Καλαμάτα, αντίστοιχα). Έχουμε υπολογίσει p(4, {2, 3, 5}) = 3 επομένως η βέλτιστη απόφαση είναι να ακολουθήσουμε τη διαδρομή 4 → 3 δηλαδή από τη Λάρισα να μεταβούμε στη Θεσσαλονίκη.

Ενώ βρισκόμαστε στην πόλη 3, οι επιλογές είναι να επισκεφτούμε είτε την 2η είτε την 5η πόλη. Το ελάχιστο κόστος συνδέεται με τη διαδρομή προς την πόλη 2 (Πάτρα) (p(3, {2, 5}) = 2). Τέλος, θα επισκεφτούμε την 5η πόλη (Καλαμάτα) πριν επιστρέψουμε και πάλι στην Αθήνα από την οποία ξεκινήσαμε. Άρα η βέλτιστη διαδρομή είναι:

1→ 4 → 3 → 2 → 5 → 1 με συνολικό κόστος 1409.

Συνοπτικά τα αποτελέσματα φαίνονται στον παρακάτω πίνακα:

ΤΡΕΧΟΥΣΑ ΠΟΛΗ	ΠΙΘΑΝΕΣ ΕΠΟΜΕΝΕΣ ΠΟΛΕΙΣ S	Si	ΕΛΑΧΙΣΤΟ ΚΟΣΤΟΣ ΔΙΑΔΡΟΜΗΣ f(I, S)	p(I, S)	ΠΡΟΣΩΡΙΝΗ ΒΕΛΤΙΣΤΗ ΔΙΑΔΡΟΜΗ
1	{2,3,4,5}	2	1645	4 5	1 --> 4 1 --> 5
		3	1426		
		4	1409		
		5	1409		
4	{2,3,5}	2	1751	3	1--> 4 --> 3
		3	1064		
		5	1756		
3	{2,5}	2	912	2	1--> 4 --> 3--> 2
		5	1150		
2	5	5	450	5	1--> 4 --> 3--> 2-->5
5	1	1	239	1	1--> 4 --> 3--> 2-->5-->1

Πίνακας 5.4.2: Συνοπτικά αποτελέσματα για εύρεση βέλτιστης διαδρομής με πρώτη μετάβαση στη Λάρισα

Περίπτωση 2: πρώτη μετάβαση ⟹ Καλαμάτα (5)

Ευρισκόμενοι στην 5η πόλη, έχουμε την επιλογή να επισκεφτούμε αμέσως μετά κάποια από τις πόλεις 2, 3 και 4 μέχρι να φτάσουμε στον τελικό προορισμό (Αθήνα). Το ελάχιστο κόστος διαδρομής εμφανίζεται κατά τη μετάβαση στην 2η πόλη (Πάτρα) οπότε επιλέγουμε αυτή ως βέλτιστη απόφαση. Από την Πάτρα οι πιθανές πόλεις που μένουν να επισκεφτούμε είναι είτε η 3 (Θεσσαλονίκη) είτε η 4 (Λάρισα). Επιλέγουμε τη μετάβαση στην πόλη 3 (p(2, {3, 4}) = 3) καθώς παρουσιάζει μικρότερο κόστος διαδρομής. Από εκεί θα ακολουθήσουμε το δρόμο προς την 4η πόλη και έπειτα θα γυρίσουμε στην πόλη από την οποία ξεκινήσαμε (Αθήνα). Έχουμε λοιπόν την εξής βέλτιστη διαδρομή:

1→ 5 → 2 → 3 → 4 → 1 με συνολικό κόστος 1409.

Συνοπτικά τα αποτελέσματα για αυτή την περίπτωση φαίνονται στον παρακάτω πίνακα:

ΤΡΕΧΟΥΣΑ ΠΟΛΗ	ΠΙΘΑΝΕΣ ΕΠΟΜΕΝΕΣ ΠΟΛΗ S	Si	ΕΛΑΧΙΣΤΟ ΚΟΣΤΟΣ ΔΙΑΔΡΟΜΗΣ f(I, S)	p(I, S)	ΠΡΟΣΩΡΙΝΗ ΒΕΛΤΙΣΤΗ ΔΙΑΔΡΟΜΗ
1	{2,3,4,5}	2	1645	4 5	1 --> 4 1 --> 5
		3	1426		
		4	1409		
		5	1409		
5	{2,3,4}	2	1170	2	1--> 5 --> 2
		3	1413		
		4	1406		
2	{3,4}	3	959	3	1--> 5 --> 2 --> 3
		4	976		
3	4	4	497	4	1--> 5 --> 2 --> 3 --> 4
4	1	1	345	1	1-->5 --> 2 --> 3 -->4-->1

Πίνακας 5.4.3: Συνοπτικά αποτελέσματα για εύρεση βέλτιστης διαδρομής με πρώτη μετάβαση στην Καλαμάτα

Παρατηρούμε ότι οι δύο βέλτιστες διαδρομές είναι αντίστροφες. Αυτό οφείλεται στη συμμετρία που παρουσιάζει το πρόβλημα. Πρακτικά, λοιπόν, συμπεραίνουμε ότι μπορούμε να ακολουθήσουμε είτε τη διαδρομή

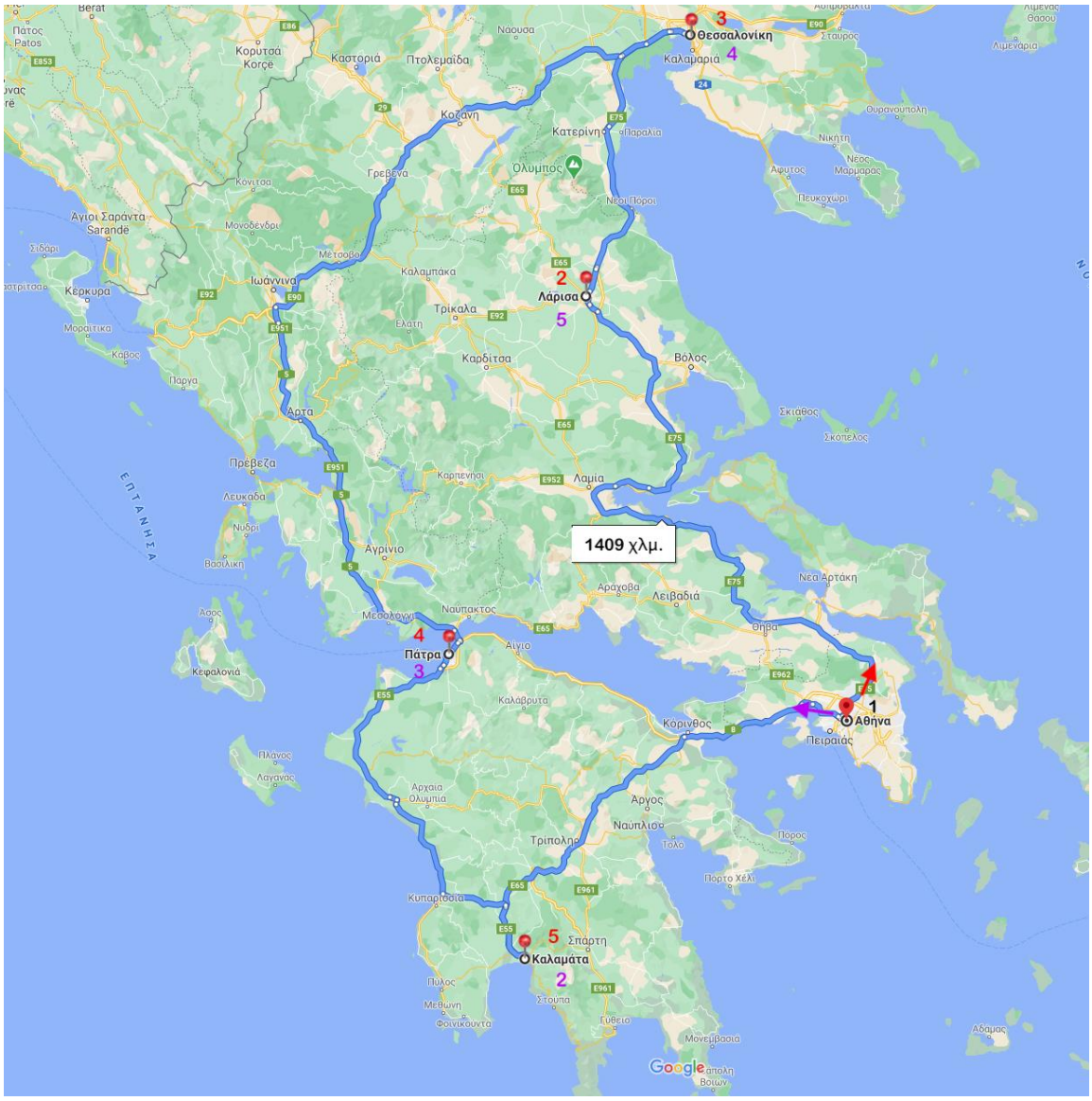
Αθήνα → Λάρισα → Θεσσαλονίκη → Πάτρα → Καλαμάτα → Αθήνα

είτε την αντίστροφη διαδρομή

Αθήνα → Καλαμάτα → Πάτρα → Θεσσαλονίκη → Λάρισα → Αθήνα

με συνολικό κόστος και στις δύο περιπτώσεις ίσο με 1409.

Η διαδρομή παρουσιάζεται στον παρακάτω χάρτη



Εικόνα 5.4.2.: Χάρτης της Ελλάδας στον οποίον απεικονίζονται οι δύο βέλτιστες διαδρομές (με κόκκινο χρώμα η πρώτη, με μωβ χρώμα η δεύτερη)

5.5. ΠΡΟΒΛΗΜΑ ΠΛΑΝΟΔΙΟΥ ΠΩΛΗΤΗ ΚΑΙ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

Όπως έχουμε ήδη αναφέρει, το πρόβλημα του πλανόδιου πωλητή θέλει πολύ χρόνο και υπολογιστικό κόστος ακόμη και με τη βοήθεια υπολογιστικών προγραμμάτων πόσο μάλλον με χαρτί και μολύβι.

Με τον αλγόριθμο Held and Karp για έναν αριθμό n κόμβων που εφαρμόζει την αναδρομική σχέση (5.1) χρειάζονται το πολύ $O(2^n n)$ επαναλήψεις, με χρόνο O(n) για την εκτέλεση της καθεμίας. Έτσι λοιπόν έχουμε ότι ο συνολικός χρόνος που χρειάζεται για να ολοκληρωθεί ο αλγόριθμος είναι $O(2^n n^2)$ με χώρο, δηλαδή μνήμη $O(2^n n)$. Προτού εφαρμόσουμε ένα ρεαλιστικό παράδειγμα σε python θα εστιάσουμε στην ερμηνεία και κατανόηση του κώδικα [39] που παρουσιάζεται στο Παράρτημα Α.

Χωρίς βλάβη της γενικότητας η πόλη 0 θεωρείται η αρχική και τελική πόλη. Για την ερμηνεία του κώδικα κρατάμε σε ισχύ όλους τους συμβολισμούς και την ερμηνεία των κεντρικών εννοιών που αναλύθηκαν στην ενότητα 5.4.

Η αναδρομική σχέση είναι:

$$f(S,i) = \begin{cases} \min_{j \in S-\{i\}} \{ (f(S\setminus\{i\}, j) + d(j,i)\}, & \alpha\nu\ |S| > 1 \\ d(1,i), & \alpha\nu\ |S| = 1 \end{cases} \qquad (5.1)$$

Η οποία δηλώνει ότι: η μικρότερη απόσταση του TSP για δοσμένο υποσύνολο πόλεων S, σε κάποιο ενδιάμεσο στάδιο που θεωρούμε ότι βρισκόμαστε στην πόλη *i* είναι ίση με τη μικρότερη απόσταση που έχουμε διανύσει πριν φτάσουμε στην πόλη i συν την απόσταση μεταξύ του τελευταίου σημείου πριν φτάσουμε στην πόλη i και της πόλης i.

Προκειμένου να μειώσουμε την πολυπλοκότητα του προβλήματος εφαρμόζουμε μία τεχνική που ονομάζεται “bitmasking” {[40], [41], [42]}. Τα bits δηλώνουν τα δυαδικά ψηφία 0 & 1 και ο όρος mask δηλώνει ότι κάτι κρύβεται. Με τον όρο bitmask γενικά εννοούμε ένα δυαδικό αριθμό ο οποίος αναπαριστά κάτι. Στο TSP θα χρησιμοποιήσουμε bitmasks ώστε να παρακολουθούμε την τρέχουσα κατάσταση του προβλήματος. Πιο συγκεκριμένα, δημιουργούμε ένα bitmask μήκους n που θα μας πληροφορεί για το ποιες πόλεις έχουμε επισκεφτεί και ποιες όχι.

Αν το bit στη θέση i είναι 0 τότε δεν έχουμε επισκεφτεί την πόλη i Αν το bit στη θέση i είναι 1 τότε έχουμε επισκεφτεί την πόλη i

Στον παρακάτω πίνακα παρουσιάζεται ένα τυχαίο παράδειγμα bitmasking ώστε να φανεί η λειτουργικότητα του στον κώδικα που χρησιμοποιούμε.

ΔΥΑΔΙΚΟΣ ΑΡΙΘΜΟΣ	1		0		1		0		
	↓		↓		↓		↓		
ΘΕΣΗ ΨΗΦΙΟΥ	3		2		1		0		
	↓		↓		↓		↓		
ΔΕΚΑΔΙΚΟΣ ΑΡΙΘΜΟΣ	2^3	+	0	+	2^1	+	0	=	10
ΣΥΝΟΛΟ S	{1,3}								

Πίνακας 5.5.1: Μετατροπή δυαδικού αριθμού σε δεκαδικό και το αντίστροφο και αναπαράσταση συνόλου S μέσω bitmasking

Δηλαδή στο παράδειγμα του πίνακα περνάμε από τις πόλεις 1 & 3 (έχοντας ως βάση την πόλη 0).

Για την εφαρμογή αυτής της τεχνικής χρησιμοποιούμε συγκεκριμένους τελεστές (bitwise operators) προκειμένου να μπορούμε να προσθέτουμε ή να αφαιρούμε δυαδικά ψηφία από τα bitmasks. Κάποιοι bitwise operators που θα μας χρειαστούν είναι οι εξής [43] :

Left Shift (x << y): επιστρέφει το x με τα bits να έχουν μετακινηθεί κατά y θέσεις αριστερά. Ισοδύναμα , x = x · 2^y

OR (x|y): θέτει κάθε bit να είναι 1 αν τουλάχιστον ένα από τα bits της ίδιας θέσης για τα x, y είναι 1

π.χ x=1101 y=1100 τότε x|y = 1101

AND (x & y): θέτει το bit να είναι 1 αν και τα 2 bits της αντίστοιχης θέσης για τα x, y είναι 1

π.χ x=1101 y=1100 τότε x&y = 1100

NOT (~x): επιστρέφει το συμπληρωματικό του x. Δηλαδή όπου 1 το 0 και αντίστροφα. Ισοδύναμα, x= -(x+1)

πχ. x= 0100 (= 4) τότε – (x+1) = -5 = - 0101 και - 0101≡ 1010

Το πλεονέκτημα της χρήσης bitmasks είναι ότι μπορούμε να προσθέτουμε το k-οστό bit στο υποσύνολο S, να το αφαιρούμε ή απλώς να ελέγχουμε αν υπάρχει μέσα στο S. Αυτό γίνεται με τη χρήση συνδυαστικών bitwise operators. Πιο συγκεκριμένα:

- Προσθήκη k-οστού bit στο υποσύνολο S και ανανέωση του S σε κάθε επανάληψη:

S | = (1<< k)

Στο TSP δηλώνει ότι η k-οστή πόλη έχει επισκεφθεί.

- Αφαίρεση k-οστού bit από το υποσύνολο S και ανανέωση του S σε κάθε επανάληψη:

  S & ~ (1<<k)

  Στο TSP δηλώνει το σύνολο S των πόλεων που έχουμε επισκεφτεί πριν φτάσουμε σε αυτή που θεωρούμε εκάστοτε τελευταία πόλη (την i).

1. Έλεγχος ύπαρξης του k-οστού bit στο υποσύνολο S (ορίζεται ως bitmask). Ορίζεται από τη σχέση:

   S & (1<< k)

   Αν το αποτέλεσμα είναι μη μηδενικό τότε το k υπάρχει στο S.

ΑΝΤΙΣΤΟΙΧΙΑ ΣΥΜΒΟΛΙΣΜΩΝ ΑΛΓΟΡΙΘΜΟΥ ΕΝΟΤΗΤΑΣ 5.4 ΚΑΙ ΚΩΔΙΚΑ

S : κάθε αυξανόμενο υποσύνολο πόλεων το οποίο δηλώνει τις πόλεις από τις οποίες περνάμε σε κάθε βήμα.

Στον κώδικα: S $\equiv bits$. Δηλαδή, αν ο (δεκαδικός) αριθμός που λαμβάνουμε σε κάθε βήμα μετατραπεί σε δυαδικό τότε μπορούμε να καταλάβουμε ποιες πόλεις περιέχονται στο σύνολο S.

i : η τρέχουσα πόλη στην οποία θέλουμε να σταματήσουμε όπου $i \in [1, \dots, n-1]$

Αναδρομική σχέση : $f(S,i) = \min_{j \in S-\{i\}} \{ (f(S\setminus\{i\}, j) + d(j,i)\}, \quad \alpha\nu\ |S| > 1$ (5.2)

Όπου,

j: η τελευταία πόλη στην οποία σταματήσαμε σε ένα προηγούμενο βήμα
$f(S,i)$: συνολικό κόστος ελάχιστης διαδρομής ξεκινώντας από την πόλη 0 σταματώντας στην πόλη i περνώντας ακριβώς μία φορά από τις πόλεις του συνόλου S
$f(S\setminus\{i\}, j)$: κόστος ελάχιστης διαδρομής προτού βρεθούμε στην πόλη i
$d(j,i)$: απόσταση μεταξύ πόλεων j και i, δηλαδή μεταξύ προηγούμενης τελευταίας πόλης j και τρέχουσας πόλης i
$p(S,i)$: η πόλη που θα επισκεφτούμε ακριβώς μετά την πόλη i, η οποία ικανοποιεί το ελάχιστο κόστος διαδρομής. Ακόμα, $p(S,i) \equiv j$ στο ελάχιστο κόστος διαδρομής.

ΣΥΜΒΟΛΙΣΜΟΙ	ΣΥΜΒΟΛΙΣΜΟΙ ΚΩΔΙΚΑ
$f(S\setminus\{i\}, j)$	$prev$
$f(S\setminus\{i\}, j) + d(j,i)$	res
$\min_{j \in S-\{i\}} \{ (f(S\setminus\{i\}, j) + d(j,i)\}$	$C = \min(res)$

Τα αποτελέσματα εμφανίζονται στην παρακάτω μορφή:

$prev$: χρησιμοποιεί τεχνική bitmasking
res: λίστα της μορφής

$$[\,(f(S\backslash\{i\},j)+d(j,i))\,,\,j\,] \Leftrightarrow [\kappa\acute{o}\sigma\tau o\varsigma, \pi\acute{o}\lambda\eta\, j]$$

Το C είναι ένα dictionary { key : value} της μορφής:

$$C\{\,(S,i) : \big(f(S,i), p(i,S)\big)\} \Leftrightarrow C\{\,(bitmask, \pi\acute{o}\lambda\eta\, i) : (\kappa\acute{o}\sigma\tau o\varsigma, \pi\acute{o}\lambda\eta\, j)\} \quad (5.3)$$

το οποίο ανανεώνεται σε κάθε βήμα.

5.5.1. ΒΗΜΑΤΑ ΑΛΓΟΡΙΘΜΟΥ HELD AND KARP ΣΕ ΚΩΔΙΚΑ

Θεωρούμε ως βάση την τιμή 0.

1. Αρχικοποιούμε το C να είναι κενό
2. Όταν k=0 το σύνολο S είναι κενό. Από αναδρομική σχέση (5.1) ισχύει ότι:
   f(S, i) = d(1, i)
3. Αποθηκεύονται στο C τα αποτελέσματα σύμφωνα με τη μορφή της σχέσης (5.3)

1η ΦΑΣΗ: θεωρώντας ως αρχή την πόλη 0 βρίσκουμε όλες τις διαδρομές μέσω της αναδρομικής σχέσης (5.2)

4. Επαναλαμβάνουμε υποσύνολα αυξανόμενου μήκους και αποθηκεύουμε τα ενδιάμεσα αποτελέσματα.
   a. Τα αυξανόμενα υποσύνολα είναι μήκους [2, ..., n-1]
   b. Επαναλαμβάνουμε τα παρακάτω βήματα για τις πόλεις που ανήκουν στο διάστημα [1, ..., n-1] και ελέγχουμε πάντα όλους τους συνδυασμούς πόλεων του εκάστοτε υποσυνόλου ανά ζεύγη.

ΥΠΟΣΥΝΟΛΑ k ΣΤΟΙΧΕΙΩΝ	ΑΥΞΑΝΟΜΕΝΑ ΥΠΟΣΥΝΟΛΑ ΠΟΛΕΩΝ
2	(1,2)
	(1,3)
	…
	(1,n)
	(2,3)
	…
	(2,n)
	(3,4)
	…
	(3, n)
3	(1,2,3)
	(1,2,4)
	…
	(1,2,n)
	(2,3,4)
	…
	(2,3,n)
4	(1,2,3,4)
	…
	(1,2,3,,n)
…	…
n	(1,2,…,n)

Πίνακας 5.5.2: Όλοι οι συνδυασμοί πόλεων των αυξανόμενων υποσυνόλων μήκους [2, …, n-1].

5. Για κάθε αυξανόμενο υποσύνολο (S) μετατρέπουμε προοδευτικά κάθε στοιχείο του σε bitmask μέχρι όλες οι πόλεις του εκάστοτε S να αναπαρίστανται μέσω ενός bitmask.

   π.χ. Για το ζεύγος (1, 2) θέλουμε το S να είναι ίσο με (1, 2) εκφρασμένο σε bitmask.
   Άρα S = $2^1 + 2^2 = 6 \Longrightarrow bitmask = 00110$[10]

6. Υπολογίζουμε το S \ { i }
   Για να συμβεί αυτό χρειάζεται να αφαιρέσουμε από το σύνολο S του βήματος 5 το στοιχείο i που θέλουμε να θεωρήσουμε ως τελευταίο.

   π.χ. Για το ζεύγος (1, 2) θεωρούμε i=1 άρα ψάχνουμε το S \ { 1 }. Εφόσον S={1, 2} μετά την αφαίρεση θα μείνει S={2}
   Μέσω bitmasking στον κώδικα θα έχουμε: prev = 4 ($\Longrightarrow$ 0100) που αντιστοιχεί στο S={2} και είναι η πόλη από την οποία περνάμε ακριβώς μία φορά όταν θεωρούμε την 1 ως τελευταία.

7. Για κάθε j $\neq i$ υπολογίζουμε το res. Στη λίστα αυτή προστίθενται κάθε φορά το αποτέλεσμα [(κόστος, j)] για κάθε j του εκάστοτε S.

8. Υπολογίζουμε το C στο οποίο προστίθενται κάθε φορά το αποτέλεσμα του min(res).

2η ΦΑΣΗ : Επιστροφή στην αρχική πόλη 0.

9. Μετατρέπουμε το S={1, 2, …, n} σε bits, ώστε να έχουμε επισκεφτεί όλες τις πόλεις από μία φορά (εκτός από την πόλη 0). Η τελική κατάσταση του υποσυνόλου S σε bits εκφράζεται ως εξής :

   $$S = (2^n - 1) - 1$$

10. Υπολογίζουμε το βέλτιστο κόστος res για το υποσύνολο S στο διάστημα [1, …, n-1] σύμφωνα με το βήμα 7 και κρατάμε το ελάχιστο κόστος διαδρομής και τη δεύτερη από την τελική πόλη 0.
11. Προχωρώντας οπισθοδρομικά βρίσκουμε τη συνολική διαδρομή
12. Αντιστρέφουμε τη διαδρομή

[10] Ο προγραμματιστής πάντα βλέπει ως αποτέλεσμα το 6 και όχι τους δυαδικούς αριθμούς. Τα bitmasks είναι εσωτερικοί τρόποι του υπολογιστή να μειώσουν το χρόνο εκτέλεσης του αλγορίθμου και η κατανόηση τους έχει να κάνει μόνο με την ερμηνεία του κώδικα.

5.5.2. ΠΑΡΑΔΕΙΓΜΑ

Από μία μεγάλη βάση δεδομένων για προβλήματα TSP παίρνουμε δεδομένα για ένα μη συμμετρικό πρόβλημα 17 κόμβων [44] (Παράρτημα Α) .
Άρα V= {0, 1, 2, …, 16} και θεωρούμε ως αρχικό και τελικό κόμβο το 0.

Εφόσον n=17 ο αριθμός των πιθανών υποσυνόλων μπορεί να είναι το πολύ $2^{17} \cdot 17 = 37{,}879{,}808$ και κάθε υποσύνολο λύνεται σε χρόνο τάξης O(17). Με άλλα λόγια ο χρόνος πολυπλοκότητας είναι $O(2^{17} \cdot 17^2)$ και ο χώρος $O(2^{17} \cdot 17)$.

Τρέχοντας τον κώδικα σύμφωνα με τον παραπάνω αλγόριθμο έχουμε τα εξής αποτελέσματα:

Βέλτιστη διαδρομή:

0→ 11→ 16→ 8→ 7→ 4→ 3→ 15→ 14→ 6→ 5→ 12→ 10→ 9→ 1→ 13→ 2 → 0

Με ελάχιστο κόστος : 39

ΚΕΦΑΛΑΙΟ 6

ΠΙΘΑΝΟΤΙΚΟΣ ΔΥΝΑΜΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΚΑΙ ΕΝΙΣΧΥΤΙΚΗ ΜΑΘΗΣΗ

6.1. ΠΙΘΑΝΟΤΙΚΟΣ ΔΥΝΑΜΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ (PROBABILISTIC DYNAMIC PROGRAMMING)

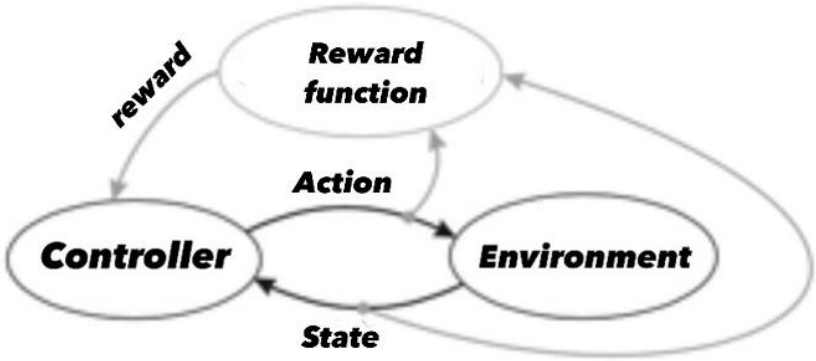
Τα προβλήματα πιθανοτικού δυναμικού προγραμματισμού ή αλλιώς στοχαστικά προβλήματα είναι αυτά των οποίων η επόμενη κατάσταση δεν καθορίζεται πλήρως από την τρέχουσα κατάσταση και απόφαση του τρέχοντος σταδίου {[1], [45]}. Δηλαδή η επόμενη κατάσταση είναι μία τυχαία μεταβλητή που δίνεται από μία κατανομή πιθανότητας. Ωστόσο, αυτή η κατανομή πιθανότητας καθορίζεται πλήρως από την τρέχουσα κατάσταση και απόφαση του τρέχοντος σταδίου, που σημαίνει ότι αυτές δίνουν την πυκνότητα πιθανότητας της τυχαίας μεταβλητής. Με άλλα λόγια, η συνάρτηση μετάβασης που έχουμε στην ντετερμινιστική περίπτωση αντικαθίσταται στη στοχαστική περίπτωση από μία πιθανότητα μετάβασης. Στο παρόν κεφάλαιο θα ασχοληθούμε με την επίλυση ενός προβλήματος πιθανοτικού δυναμικού προγραμματισμού που εκτείνεται σε ένα ευρύτερο πλαίσιο αλγορίθμων και μεθόδων που ονομάζεται Ενισχυτική Μάθηση. Θα εστιάσουμε σε διακριτούς[11] χώρους καταστάσεων προκειμένου να βρεθούν ακριβείς βέλτιστες λύσεις. Για συνεχείς[12] ή μεγάλους χώρους καταστάσεων απαιτούνται προσεγγιστικές μέθοδοι με τις οποίες δεν θα ασχοληθούμε στην παρούσα εργασία.

6.2. ΕΝΙΣΧΥΤΙΚΗ ΜΑΘΗΣΗ (REINFORCEMENT LEARNING)

Η Ενισχυτική Μάθηση (Reinforcement Learning - RL) όπως και ο Δυναμικός Προγραμματισμός [45] είναι αλγοριθμικές μέθοδοι επίλυσης προβλημάτων στα οποία παίρνονται αποφάσεις για ένα σύστημα σε ένα εκτεταμένο χρονικό πλαίσιο για την επίτευξη ενός στόχου. Πιο συγκεκριμένα, η ενισχυτική μάθηση σε ένα στοχαστικό πλαίσιο είναι η διαδικασία εκπαίδευσης ενός χειριστή μέσω επιβραβεύσεων ή τιμωριών σε ένα περίπλοκο, αβέβαιο περιβάλλον για την επίτευξη ενός στόχου. Οι αλγόριθμοι του RL χρησιμοποιούνται για προβλήματα που δεν περιγράφονται από ένα συγκεκριμένο μοντέλο (model free) είτε γιατί αυτό δεν είναι γνωστό, είτε γιατί δεν είναι επαρκώς κατανοητό είτε γιατί η δημιουργία ενός ικανοποιητικού μοντέλου είναι δύσκολη ή έχει μεγάλο υπολογιστικό κόστος. Αυτό σημαίνει ότι το περιβάλλον που μελετάται δεν περιγράφεται από καμία γνωστή κατανομή πιθανότητας. Το πλεονέκτημα του RL είναι ότι μπορεί να λειτουργήσει χρησιμοποιώντας μόνο δεδομένα που λαμβάνονται από το περιβάλλον που μελετάται. Κατηγοριοποιούνται σε μεθόδους “offline” [45], όταν τα δεδομένα λαμβάνονται εκ των προτέρων και μεθόδους “online” [45] όταν δεν υπάρχουν διαθέσιμα δεδομένα και το περιβάλλον αλληλοεπιδρά με τον χρήστη προκειμένου να βρει λύση. Αντίθετα, οι αλγόριθμοι του δυναμικού προγραμματισμού λύνουν προβλήματα των οποίων το περιβάλλον περιγράφεται πλήρως από ένα γνωστό μοντέλο. Με άλλα λόγια, οι κατανομές πιθανότητας οποιασδήποτε αλλαγής

[11] Ο αριθμός των καταστάσεων στο χώρο καταστάσεων είναι πεπερασμένος [45].
[12] Ο αριθμός των καταστάσεων στο χώρο καταστάσεων μπορεί να είναι άπειρος [45].

που συμβαίνει στη ρύθμιση του προβλήματος είναι γνωστές. Αυτό ακριβώς είναι και το βασικό χαρακτηριστικό που διαχωρίζει τους DP και RL. Με μία πιο άτυπη προσέγγιση θα μπορούσαμε να πούμε ότι ο δυναμικός προγραμματισμός είναι ένα υποσύνολο της ενισχυτικής μάθησης το οποίο περιορίζεται σε προβλήματα βασισμένα σε γνωστά μοντέλα ταυτόχρονα όμως αποτελεί και τη βάση για την εισαγωγή και εξέλιξη αυτού του τομέα. Ο στόχος τους είναι κοινός και δεν είναι άλλος από την εύρεση μίας βέλτιστης πολιτικής που θα μεγιστοποιεί την αναμενόμενη[13] απόδοση η οποία λαμβάνεται μέσω μίας αθροιστικής ανταμοιβής/επιβράβευσης κατά τη διάρκεια αλληλεπίδρασης του χειριστή και του περιβάλλοντος. Η γενική διαδικασία με την οποία λειτουργεί ένα σύστημα Ενισχυτικής Μάθησης είναι ίδια με την αναλυτική περιγραφή που δώσαμε στην ενότητα 2.1. για ένα σύστημα δυναμικού προγραμματισμού. Γι' αυτό παραθέτουμε ξανά μόνο το σχήμα που παρουσιάζει μία κυκλική διαδικασία όπως παρουσιάστηκε στην προαναφερθείσα ενότητα:



Σχήμα 6.1: Κυκλική διαδικασία αλληλεπίδρασης που ακολουθείται σε DP και RL (Busoniu, Babuska, De Schutter, & Ernst, 2010)

Και οι δύο «συλλογές» αλγορίθμων χωρίζονται σε τρεις υποκατηγορίες αλγορίθμων αναλόγως τον τρόπο που επιλέγουν να βρουν μία βέλτιστη πολιτική [45]:

1. Αλγόριθμοι Επανάληψης Τιμών (Value Iteration Algorithms): ψάχνουν την βέλτιστη συνάρτηση τιμής (value function) η οποία αποτελείται από τις μέγιστες αποδόσεις κάθε κατάστασης ή από κάθε ζεύγος κατάστασης – απόφασης. Η βέλτιστη συνάρτηση χρησιμοποιείται για την εύρεση της βέλτιστης πολιτικής.

2. Αλγόριθμοι Επανάληψης Πολιτικής (Policy Iteration Algorithms): αξιολογούν τις διάφορες πολιτικές κατασκευάζοντας τις συναρτήσεις τιμών τους και στη συνέχεια χρησιμοποιούν τις συναρτήσεις αυτές ώστε να βρουν νέες, βελτιωμένες πολιτικές.

3. Αλγόριθμοι Εύρεσης Πολιτικής (Policy Search Algorithms): ψάχνουν απευθείας μία βέλτιστη πολιτική εφαρμόζοντας τεχνικές βελτιστοποίησης.

[13] Αναφερόμαστε στη στοχαστική περίπτωση

Η ενισχυτική μάθηση σήμερα βρίσκεται σε συνεχή εξέλιξη και εμφανίζεται σε πολλούς τομείς από την επιχειρησιακή έρευνα και τη χημεία μέχρι την τεχνητή νοημοσύνη και τη ρομποτική [46]. Ακόμη, πλήθος ηλεκτρονικών παιχνιδιών βασίζονται σε αλγορίθμους RL. Μία από τις μεγαλύτερες επιτυχίες της ενισχυτικής μάθησης μέχρι σήμερα στον τομέα των παιχνιδιών αφορά στο παιχνίδι Go. Τα προγράμματα της Google "AlphaGo" και "AlphaGo Zero" ήταν εκείνα που κατάφεραν αποτελεσματικά να εκπαιδεύσουν το χειριστή να αποκτήσει υπεράνθρωπες ικανότητες που οδήγησαν στο να κερδίζει τους καλύτερους επαγγελματίες παίκτες στον κόσμο {[67], [68]}. Όπως καταλαβαίνουμε, η ενισχυτική μάθηση δεν είναι εύκολη υπόθεση και οι αλγόριθμοι χρειάζονται ένα περιβάλλον στο οποίο μπορούν να ελέγχονται και να συγκρίνονται με άλλους διαφορετικούς αλγορίθμους RL. Μία βιβλιοθήκη με πολλά περιβάλλοντα είναι το OpenAI Gym μέσα στο οποίο βρίσκουμε και το παράδειγμα το οποίο θα αναλύσουμε παρακάτω.

6.3. OPEN AI

Το OpenAI {[47], [48]} είναι ένα ερευνητικό εργαστήριο τεχνητής νοημοσύνης που ιδρύθηκε το 2015 με στόχο την προώθηση και την εξέλιξη της τεχνητής νοημοσύνης. Με βάση μία έρευνα που πραγματοποιήθηκε το 2016 [49] μεγαλύτερο από το 70% των ερευνητών προσπάθησαν και απέτυχαν να αναπαραγάγουν πειράματα ενός άλλου ερευνητή ενώ περισσότεροι από τους μισούς απέτυχαν να αναπαραγάγουν τη δική τους δουλεία. Η ίδια έρευνα έδειξε ότι κάποιοι από τους βασικότερους λόγους που οδηγούν σε αυτή την αποτυχία αναπαραγωγισιμότητας είναι η έλλειψη πειραματικού σχεδιασμού και η περιορισμένη διαθεσιμότητα τόσο σε μεθόδους και κώδικες όσο και σε δεδομένα από κάποιο πρότυπο εργαστήριο. Η δημιουργία του OpenAI έχει ως στόχο την εξάλειψη αυτού του προβλήματος με τη δημιουργία μεγάλου αριθμού περιβαλλόντων στα οποία θα υπάρχουν κοινόχρηστες διασυνδέσεις για την ανάπτυξη και τον έλεγχο αλγορίθμων. Με αυτό τον τρόπο θα αυξηθεί η αναπαραγωγισιμότητα στον τομέα της τεχνητής νοημοσύνης που σημαίνει ακόμα μεγαλύτερη εξέλιξη τα επόμενα χρόνια.

Το OpenAI Gym {[50], [51]} είναι μία κατασκευή του OpenAI, μία εργαλειοθήκη, για την ανάπτυξη και σύγκριση διαφόρων αλγόριθμων RL. Διαθέτει ένα μεγάλο αριθμό εύχρηστων περιβαλλόντων όπου ένας προγραμματιστής μπορεί να χρησιμοποιήσει προκειμένου να δουλέψει δικούς του αλγορίθμους ή να αναπαραγάγει άλλους διαθέσιμους προκειμένου να μελετήσει ένα συγκεκριμένο πρόβλημα. Μέχρι στιγμής, το OpenAI Gym χρησιμοποιείται μόνο με γλώσσα Python.

Στο στοχαστικό παράδειγμα που θα παρουσιάσουμε παρακάτω θα χρησιμοποιήσουμε το περιβάλλον του OpenAI Gym. Ωστόσο, προκειμένου να περιγραφεί πλήρως θα κάνουμε μία μικρή εισαγωγή στις Μαρκοβιανές Διαδικασίες Απόφασης και στη συμβολή τους στην επίλυση προβλημάτων απόφασης ενισχυτικής μάθησης και δυναμικού προγραμματισμού που πληρούν συγκεκριμένα χαρακτηριστικά.

6.4. ΜΑΡΚΟΒΙΑΝΕΣ ΔΙΑΔΙΚΑΣΙΕΣ ΑΠΟΦΑΣΗΣ ΣΤΗΝ ΕΝΙΣΧΥΤΙΚΗ ΜΑΘΗΣΗ

Υπενθυμίζουμε ότι τα βασικά στοιχεία ενός προβλήματος ενισχυτικής μάθησης και δυναμικού προγραμματισμού είναι ο χειριστής, το περιβάλλον, ο χώρος καταστάσεων (X), ο χώρος αποφάσεων (U), η επιβράβευση (R) και η πιθανότητα μετάβασης [60]. Οι Μαρκοβιανές διαδικασίες {[52], [54]} αποτελούν ένα πλαίσιο μέσα στο οποίο μπορούν να λυθούν πολλά προβλήματα RL και DP που περιλαμβάνουν διακριτές αποφάσεις. Όταν τα προβλήματα αυτά ικανοποιούν τη λεγόμενη αρχή "Markov Property" [69], στην οποία ορίζεται ότι μία μελλοντική κατάσταση εξαρτάται μόνο από το παρόν, δηλαδή από την τρέχουσα κατάσταση και απόφαση, και όχι από το παρελθόν, τότε μπορούν να οριστούν μέσω μίας Μαρκοβιανής διαδικασίας απόφασης (MDP). Με αυτό τον τρόπο βοηθούν το χειριστή να φτάσει σε μία βέλτιστη πολιτική με μέγιστες επιβραβεύσεις με την πάροδο του χρόνου. Με τον όρο βέλτιστη πολιτική εννοούμε την πολιτική που μεγιστοποιεί τη συνολική αθροιστική επιβράβευση. Για το σκοπό αυτό χρειάζονται δύο βασικά στοιχεία:

1. Ένας τρόπος ώστε να αποφασίζεται η αξία της εκάστοτε κατάστασης
2. Μία αναμενόμενη τιμή που να προσδιορίζει την αξία της απόφασης που λαμβάνεται σε μία συγκεκριμένη κατάσταση

Με βάση λοιπόν τα 1 και 2 ορίζουμε δύο συναρτήσεις τιμών, τις λεγόμενες "V-function" και "Q-function", αντίστοιχα.

Η συνάρτηση-V (V-function) {[45], [54]} μίας πολιτικής είναι γνωστή ως συνάρτηση κατάστασης (state-value function) καθώς υπολογίζει τη συνολική απόδοση[14] που περιμένουμε ξεκινώντας από μία κατάσταση x ακολουθώντας μία πολιτική h. Με άλλα λόγια υπολογίζει πόσο καλό είναι για τον χειριστή να βρίσκεται σε μία δοθείσα κατάσταση. Η βέλτιστη συνάρτηση V είναι η καλύτερη συνάρτηση τιμής ακολουθώντας κάποια πολιτική και συμβολίζεται με V^*. Η βέλτιστη αυτή τιμή υπολογίζεται μέσω της «εξίσωσης Bellman» {[45], [53]} μίας εξίσωσης που αποτελεί τη βάση της ενισχυτικής μάθησης και του δυναμικού προγραμματισμού.

Εξίσωση Bellman για τη συνάρτηση V^*:

$$V^*(x) = \max_u \{R(x,u,x') + \gamma \cdot \sum_{x'} P(x'|x,u) \cdot V(x')\} \qquad (6.1)$$

Όπου,

$R(x,u,x')$: *άμεση αναμενόμενη επιβράβευση από τη μετάβαση της κατάστασης* x *στην επόμενη κατάσταση* x' *μέσω της απόφασης* u.

$P(x'|x,u)$: *πιθανότητα μετάβασης από την κατάσταση* x *στην κατάσταση* x' *όπου ο χειριστής λαμβάνει μία απόφαση* u.

[14] Το άθροισμα των επιβραβεύσεων που δέχεται ο χειριστής από τη στιγμή που ξεκινάει η διαδικασία στην αρχική κατάσταση μέχρι μία τελική κατάσταση όπου η διαδικασία τερματίζεται.

γ $(discount\ factor)$ [54]: *συντελεστής* $(\gamma \in [0,1])$ *που μετράει κατά πόσο ο χειριστής δίνει αξία στις επιβραβεύσεις με την πάροδο του χρόνου. Όσο πιο κοντά βρίσκεται στη μονάδα τόσο μικρότερη αξία έχουν οι επιβραβεύσεις στην τρέχουσα κατάσταση.*

Ωστόσο, η βέλτιστη τιμή κατάστασης δεν παρέχει στο χειριστή την πληροφορία για το ποια απόφαση να λάβει στην εκάστοτε κατάσταση. Για το λόγο αυτό εισάγεται η «Συνάρτηση – Q» ή αλλιώς «συνάρτηση-απόφασης» (action-value function).

Η συνάρτηση-Q (Q-function) {[45], [54]} καθορίζει την αξία μίας ληφθείσας απόφασης στην εκάστοτε κατάσταση x ακολουθώντας μία πολιτική h, δηλαδή υπολογίζει πόσο καλή ή κακή είναι η λήψη μίας συγκεκριμένης απόφασης στην εκάστοτε τρέχουσα κατάσταση. Αυτή η προσέγγιση που δίνει έμφαση στην ποιότητα της απόφασης είναι γνωστή ως *Q-Learning [56].*

Η αντίστοιχη εξίσωση Bellman [56] για τη βέλτιστη συνάρτηση Q^*:

$$Q^*(x,u) = R(x,u) + \gamma \cdot \sum_{x'} P(x'|x,u) \cdot V(x') \qquad (6.2)$$

Ισχύει ότι $V(x) = maxQ(x,u)$ *άρα η τελική μορφή της συνάρτησης Q είναι η εξής:*

$$Q^*(x,u) = R(x,u) + \gamma \cdot \sum_{x'} P(x'|x,u) \cdot \max_{u'} Q^*(x',u') \qquad (6.3)$$

Δηλαδή, η βέλτιστη εξίσωση Bellman (6.3) δηλώνει ότι η βέλτιστη τιμή της απόφασης u που λαμβάνεται στην κατάσταση x ισούται με την αναμενόμενη άμεση επιβράβευση συν τη βέλτιστη τιμή που λαμβάνεται από την καλύτερη απόφαση u' της επόμενης κατάστασης x' μειωμένη κατά τον συντελεστή γ. Αυτή η μείωση οφείλεται στο χαρακτηριστικό του γ, όσο ισχύει γ<1, ότι το να λαμβάνει ο χειριστής επιβράβευση στο μέλλον έχει μικρότερη αξία από το να λαμβάνει την ίδια επιβράβευση άμεσα.

Ο χειριστής επαναλαμβάνει την ίδια διαδικασία εύρεσης Q^* για κάθε κατάσταση επιλέγοντας μία απόφαση, ακολουθώντας σε κάθε επανάληψη μία πολιτική. Σε κάθε επόμενη επανάληψη υπολογίζονται νέες τιμές $Q^*(x,u)$ για κάθε κατάσταση, μέσω της σχέσης (6.3), μέχρι ως ότου η διαφορά δύο διαδοχικών τιμών να είναι μικρότερη από έναν μικρό αριθμό ε > 0, δηλαδή μέχρι η διαδικασία να συγκλίνει. Η διαφορά αυτή ονομάζεται χρονική διαφορά (temporal difference-TD) {[55], [57]} και υπολογίζεται ως εξής:

$$TD(x,u) = newQ^*(x,u) - oldQ^*(x,u) \qquad (6.5)$$

Η νέα τιμή της $Q^*(x,u)$ είναι:

$$Q_t^*(x,u) = Q^*_{t-1}(x,u) + \alpha \cdot TD_t(x,u) \Rightarrow$$

$$Q_t^*(x,u) = Q^*_{t-1}(x,u) + \alpha \cdot \left(R(x,u) + \gamma \cdot \max_{u'} Q^*(x',u') - Q^*_{t-1}(x,u)\right) \quad (6.6)$$

όπου α: συντελεστής εκμάθησης [56] $(0 < a \leq 1)$ ο οποίος ελέγχει πόσο γρήγορα ο χειριστής προσαρμόζεται στις τυχαίες αλλαγές που συμβαίνουν στο περιβάλλον

Μόλις, ικανοποιηθεί η σύγκλιση θα έχουμε βρει τη βέλτιστη πολιτική h^* η οποία μεγιστοποιεί τη βέλτιστη συνάρτηση-Q^* και ισχύει:

$$h^*(x) \in arg\max_u Q^*(x,u) \quad (6.7)$$

6.5. ΤΟ ΠΡΟΒΛΗΜΑ ΤΗΣ ΠΑΓΩΜΕΝΗΣ ΛΙΜΝΗΣ

Δημιουργούμε ένα παιχνίδι [58] κατά το οποίο ένα ρομπότ βρισκόμενο σε μία παγωμένη λίμνη προσπαθεί να φτάσει ένα αντικείμενο. Σε κάποια σημεία ο πάγος έχει λιώσει με αποτέλεσμα να υπάρχουν τρύπες στην επιφάνεια. Ακόμη, λόγω ολισθηρότητας της επιφάνειας, η εκάστοτε κίνηση του ρομπότ είναι αβέβαιη, δηλαδή μπορεί να μην μετακινείται πάντα προς την κατεύθυνση που έχει βάλει στόχο. Το παιχνίδι τερματίζεται όταν το ρομπότ πέσει σε μία τρύπα ή φτάσει στον τελικό στόχο, δηλαδή στη θέση του αντικειμένου. Αν φτάσει επιτυχώς στο στόχο η επιβράβευση έχει αξία 1 ενώ σε οποιαδήποτε άλλη θέση δεν υπάρχει επιβράβευση, δηλαδή είναι ίση με 0.

Στόχος λοιπόν είναι μέσω πολλών επαναλήψεων των αλγορίθμων της ενισχυτικής μάθησης [59] να εκπαιδεύσουμε το ρομπότ να φτάνει στον τελικό στόχο όσο το δυνατόν γρηγορότερα χωρίς να πέφτει σε κάποια τρύπα.

Έχουμε λοιπόν τα εξής δεδομένα:

Χειριστής: ρομπότ
Περιβάλλον: επιφάνεια λίμνης

Η επιφάνεια της λίμνης που αναπαριστά το περιβάλλον του προβλήματος παρουσιάζεται στο παρακάτω σχήμα:

S	F	F	F
F	H	F	H
F	F	F	H
H	F	F	G

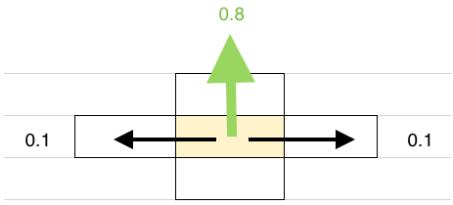
Πίνακας 6.5.1: Αναπαράσταση περιβάλλοντος

Όπου:

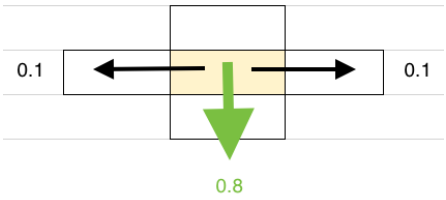
S: αρχική κατάσταση, ασφαλής θέση
F: παγωμένη επιφάνεια, ασφαλής θέση
H: τρύπα, τερματίζεται η διαδικασία με επιβράβευση 0
G: στόχος- εύρεση αντικειμένου, τερματίζεται η διαδικασία με επιβράβευση 1

Κάθε κελί του πίνακα παρουσιάζει μία κατάσταση x, συνεπώς υπάρχουν 16 πιθανές καταστάσεις, δηλαδή 16 θέσεις στις οποίες μπορεί να κινηθεί το ρομπότ. Θεωρούμε, λοιπόν, ως χώρο καταστάσεων τον X = {[1, 1], [1, 2], [1, 3], [1, 4], [2, 1], [2, 2], [2, 3], [2, 4], [3, 1], [3, 2], [3, 3], [3, 4], [4, 1], [4, 2], [4, 3], [4, 4]}.Για κάθε κατάσταση οι αποφάσεις/ μετακινήσεις u είναι 4 (πάνω (U), κάτω (D), αριστερά (L), δεξιά (R)), δηλαδή το ρομπότ μπορεί να μετακινηθεί σε μία από αυτές τις κατευθύνσεις με πιθανότητα 0.8 να φτάσει στη θέση για την οποία

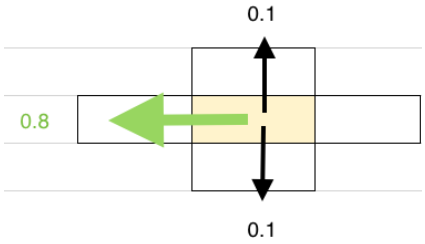
ξεκίνησε και 0.1 αντίστοιχα για να βρεθεί είτε δεξιά είτε αριστερά από την προκαθορισμένη θέση. Επομένως, ο χώρος αποφάσεων είναι U={0, 1, 2, 3} όπου 0: αριστερά, 1: κάτω, 2: δεξιά, 3: πάνω. Η συνάρτηση επιβράβευσης συμβολίζεται με R και μπορεί να είναι 0 ή 1. Στην περίπτωση που η κίνηση του ρομπότ πάει να βγει εκτός ορίων τότε θεωρούμε ότι παραμένει στην ίδια θέση.
Οι πιθανότητες για κάθε πιθανή μετακίνηση του ρομπότ φαίνονται στα παρακάτω σχήματα.



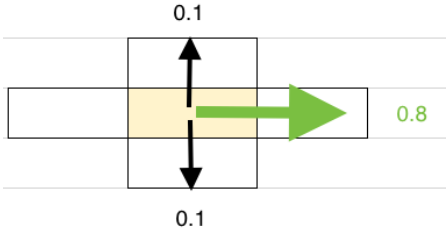
Σχήμα 6.5.1: Το ρομπότ κατευθύνεται προς τα πάνω με πιθανότητα 0.8 και με πιθανότητα 0.1 σε κάθε πλευρά αντίστοιχα.



Σχήμα 6.5.2: Το ρομπότ κατευθύνεται προς τα κάτω με πιθανότητα 0.8 και με πιθανότητα 0.1 σε κάθε πλευρά αντίστοιχα.



Σχήμα 6.5.3: Το ρομπότ κατευθύνεται προς τα αριστερά με πιθανότητα 0.8 και με πιθανότητα 0.1 σε κάθε κατακόρυφη θέση αντίστοιχα.



Σχήμα 6.5.4: Το ρομπότ κατευθύνεται προς τα δεξιά με πιθανότητα 0.8 και με πιθανότητα 0.1 σε κάθε κατακόρυφη θέση αντίστοιχα.

Η εφαρμογή του αλγορίθμου για την εύρεση βέλτιστης λύσης στηρίζεται στην εξίσωση Bellman. Θεωρούμε γ=0.1 και η αναδρομική σχέση (6.3) έχει τη μορφή:

$$Q^*(x,u) = R(x,u) + 0.1 \cdot \sum_{x'} P(x'|x,u) \cdot \max_{u'} Q^*(x',u') \quad (6.8)$$

Η συνάρτηση επιβράβευσης:

$$R(x,u,x') = \begin{cases} 1, & \alpha\nu\ x \neq [4,4]\ \kappa\alpha\iota\ x' = [4,4] \\ 0, & \alpha\lambda\lambda\iota\acute{\omega}\varsigma \end{cases} \quad (6.9)$$

Θα παρουσιάσουμε μερικά αρχικά τυχαία βήματα του ρομπότ μέχρι να ολοκληρωθούν δύο επεισόδια [54]. Ένα επεισόδιο θεωρείται από την αρχική κατάσταση του ρομπότ μέχρι να βρεθεί σε μία τερματική κατάσταση, δηλαδή μέχρι να πέσει σε τρύπα ή να φτάσει στο αντικείμενο.

Παρακάτω δίνεται ο πίνακας των επιβραβεύσεων για κάθε απόφαση σε κάθε κατάσταση, όπου κάθε γραμμή απεικονίζει μία κατάσταση και κάθε στήλη μία απόφαση, δηλαδή είναι ένας πίνακας 16x4:

	Αποφάσεις	αριστερά (L)	κάτω (D)	δεξιά (R)	πάνω (U)
Καταστάσεις	[1,1]	0	0	0	0
	[1,2]	0	0	0	0
	[1,3]	0	0	0	0
	[1,4]	0	0	0	0
	[2,1]	0	0	0	0
	[2,2]	0	0	0	0
	[2,3]	0	0	0	0
	[2,4]	0	0	0	0
	[3,1]	0	0	0	0
	[3,2]	0	0	0	0
	[3,3]	0	0	0	0
	[3,4]	0	0	0	0
	[4,1]	0	0	0	0
	[4,2]	0	0	0	0
	[4,3]	0	0	1	0
	[4,4]	0	0	0	0

Πίνακας 6.5.2: Πίνακας επιβράβευσης. Επιβράβευση 1 για τον τελικό στόχο και 0 σε κάθε άλλη περίπτωση

Θα δημιουργήσουμε έναν πίνακα Q^* ίδιου μεγέθους στον οποίο θα αποθηκεύονται οι τιμές $Q^*(x,u)$ για κάθε απόφαση του ρομπότ σε κάθε κατάσταση. Αρχικοποιούμε τον πίνακα Q^* να είναι μηδενικός.

	Αποφάσεις	αριστερά (L)	κάτω (D)	δεξιά (R)	πάνω (U)
Καταστάσεις	[1,1]	0	0	0	0
	[1,2]	0	0	0	0
	[1,3]	0	0	0	0
	[1,4]	0	0	0	0
	[2,1]	0	0	0	0
	[2,2]	0	0	0	0
	[2,3]	0	0	0	0
	[2,4]	0	0	0	0
	[3,1]	0	0	0	0
	[3,2]	0	0	0	0
	[3,3]	0	0	0	0
	[3,4]	0	0	0	0
	[4,1]	0	0	0	0
	[4,2]	0	0	0	0
	[4,3]	0	0	0	0
	[4,4]	0	0	0	0

Πίνακας 6.5.3: Αρχικός πίνακας Q (μηδενικός)

Η σειρά με την οποία το ρομπότ επισκέπτεται την εκάστοτε κατάσταση είναι τυχαία αλλά πρέπει μετά από Ν αριθμό επαναλήψεων να έχει βρεθεί σε όλες τις καταστάσεις έχοντας επιλέξει κάθε δυνατή απόφαση.

ΑΡΧΗ ΔΙΑΔΙΚΑΣΙΑΣ

Έστω ότι το ρομπότ βρίσκεται τυχαία στην κατάσταση x=[4, 3]. Το ρομπότ έχει την επιλογή να μετακινηθεί πάνω, κάτω, αριστερά και δεξιά. Ωστόσο αν μετακινηθεί κάτω θα βρει τοίχο οπότε θα θεωρούμε ότι για αυτή τη μετακίνηση παραμένει στη θέση του. Επιλέγεται τυχαία να μετακινηθεί δεξιά. Με βάση αυτή την απόφαση μεταβαίνει με πιθανότητα 0.8 στην κατάσταση δεξιά (x=[4, 4]), δηλαδή σε εκείνη για την οποία ξεκίνησε και με πιθανότητα 0.1 στην πάνω (x=[3, 3]) και 0.1 στην ίδια κατάσταση (x=[4,3]). Εφαρμόζοντας την εξίσωση (6.3) η συνάρτηση-Q^* είναι της μορφής:

$$Q^*([4,3],R) = R([4,3],R) + 0.1 \cdot \{0.8 \cdot maxQ^*([4,4],u') + 0.1 \cdot maxQ^*([3,3],u') + 0.1 \cdot maxQ^*([4,3],u')\} = 1 + 0.1 \cdot \{0.8 \cdot 0 + 0.1 \cdot 0 + 0.1 \cdot 0\} = 1$$

Τώρα το ρομπότ βρίσκεται στην κατάσταση x=[4, 4]. Η θέση αυτή είναι ο τελικός στόχος οπότε το ρομπότ επιβραβεύεται με 1 και το επεισόδιο τερματίζεται.
Ο ανανεωμένος πίνακας έχει την εξής μορφή:

	Αποφάσεις	αριστερά (L)	κάτω (D)	δεξιά (R)	πάνω (U)
Καταστάσεις	[1,1]	0	0	0	0
	[1,2]	0	0	0	0
	[1,3]	0	0	0	0
	[1,4]	0	0	0	0
	[2,1]	0	0	0	0
	[2,2]	0	0	0	0
	[2,3]	0	0	0	0
	[2,4]	0	0	0	0
	[3,1]	0	0	0	0
	[3,2]	0	0	0	0
	[3,3]	0	0	0	0
	[3,4]	0	0	0	0
	[4,1]	0	0	0	0
	[4,2]	0	0	0	0
	[4,3]	0	0	1	0
	[4,4]	0	0	0	0

Πίνακας 6.5.4: Ανανεωμένος πίνακας Q^* όταν το ρομπότ έφτασε στο στόχο

Για το επόμενο επεισόδιο θεωρούμε τυχαία ως αρχική κατάσταση την x=[4, 2]. Επιλέγεται πάλι τυχαία το ρομπότ να μετακινηθεί προς τα δεξιά. Συνεπώς μεταβαίνει στην κατάσταση x=[4, 3] με πιθανότητα 0.8, στην κατάσταση x=[3, 2] και στην ίδια θέση με πιθανότητα 0.1 και 0.1, αντίστοιχα. Η συνάρτηση-Q^* υπολογίζεται:

$Q^*([4,2],R) = R([4,2],R) + 0.1 \cdot \{0.8 \cdot maxQ^*([4,3],u') + 0.1 \cdot maxQ^*([3,2],u') + 0.1 \cdot maxQ^*([4,2]) = 0 + 0.1 \cdot \{0.8 \cdot 1 + 0.1 \cdot 0 + 0.1 \cdot 0\} = 0.08$

Ο ανανεωμένος πίνακας-Q* είναι:

	Αποφάσεις	αριστερά (L)	κάτω (D)	δεξιά (R)	πάνω (U)
Καταστάσεις	[1,1]	0	0	0	0
	[1,2]	0	0	0	0
	[1,3]	0	0	0	0
	[1,4]	0	0	0	0
	[2,1]	0	0	0	0
	[2,2]	0	0	0	0
	[2,3]	0	0	0	0
	[2,4]	0	0	0	0
	[3,1]	0	0	0	0
	[3,2]	0	0	0	0
	[3,3]	0	0	0	0
	[3,4]	0	0	0	0
	[4,1]	0	0	0	0
	[4,2]	0	0	0.08	0
	[4,3]	0	0	1	0
	[4,4]	0	0	0	0

Πίνακας 6.5.5: Ανανεωμένος πίνακας Q μετά τη δεξιά μετακίνηση του ρομπότ από την κατάσταση [4,2]*

Το ρομπότ βρίσκεται τώρα στην κατάσταση x=[4,3]. Τυχαία επιλέγεται να μετακινηθεί προς τα πάνω. Με ανάλογο τρόπο η συνάρτηση-Q* υπολογίζεται:

$Q^*([4,3],U) = R([4,3],U) + 0.1 \cdot \{0.8 \cdot maxQ^*([3,3],u') + 0.1 \cdot maxQ^*([4,2],u') + 0.1 \cdot maxQ^*([4,4],u')\} = 0 + 0.1 \cdot \{0.8 \cdot 0 + 0.1 \cdot 0.08 + 0.1 \cdot 0\} = 0.1 \cdot 0.008 = 0.0008$

Ο ανανεωμένος πίνακας-Q* είναι:

	Αποφάσεις	αριστερά (L)	κάτω (D)	δεξιά (R)	πάνω (U)
Καταστάσεις	[1,1]	0	0	0	0
	[1,2]	0	0	0	0
	[1,3]	0	0	0	0
	[1,4]	0	0	0	0
	[2,1]	0	0	0	0
	[2,2]	0	0	0	0
	[2,3]	0	0	0	0
	[2,4]	0	0	0	0
	[3,1]	0	0	0	0
	[3,2]	0	0	0	0
	[3,3]	0	0	0	0
	[3,4]	0	0	0	0
	[4,1]	0	0	0	0
	[4,2]	0	0	0.08	0
	[4,3]	0	0	1	0.0008
	[4,4]	0	0	0	0

Πίνακας 6.5.6: Ανανεωμένος πίνακας Q^* μετά την προς τα πάνω μετακίνηση του ρομπότ από την κατάσταση [4,3]

Το ρομπότ βρίσκεται στην κατάσταση x=[3, 3]. Επιλέγεται τυχαία να μετακινηθεί αριστερά.

$$Q^*([3,3],L) = R([3,3],L) + 0.1 \cdot \{0.8 \cdot maxQ^*([3,2] + 0.1 \cdot maxQ^*([4,3],u') + 0.1 \cdot maxQ^*([2,3],u')\} = 0 + 0.1 \cdot \{0.8 \cdot 0 + 0.1 \cdot 1 + 0.1 \cdot 0\} = 0.1 \cdot 0.1 = 0.01$$

Ο ανανεωμένος πίνακας-Q^* είναι:

	Αποφάσεις	αριστερά (L)	κάτω (D)	δεξιά (R)	πάνω (U)
Καταστάσεις	[1,1]	0	0	0	0
	[1,2]	0	0	0	0
	[1,3]	0	0	0	0
	[1,4]	0	0	0	0
	[2,1]	0	0	0	0
	[2,2]	0	0	0	0
	[2,3]	0	0	0	0
	[2,4]	0	0	0	0
	[3,1]	0	0	0	0
	[3,2]	0	0	0	0
	[3,3]	0.01	0	0	0
	[3,4]	0	0	0	0
	[4,1]	0	0	0	0
	[4,2]	0	0	0.08	0
	[4,3]	0	0	1	0.0008
	[4,4]	0	0	0	0

Πίνακας 6.5.7: Ανανεωμένος πίνακας Q^* μετά την αριστερή μετακίνηση του ρομπότ από την την κατάσταση [3, 3]

Το ρομπότ βρίσκεται στην κατάσταση x=[3,2]. Επιλέγεται τυχαία να μετακινηθεί προς τα κάτω. Η συνάρτηση-Q* υπολογίζεται:

$$Q^*([3,2],D) = R([3,2],D) + 0.1 \cdot \{0.8 \cdot maxQ^*([4,2] + 0.1 \cdot maxQ^*([3,3],u') + 0.1 \cdot maxQ^*([3,1],u')\} = 0 + 0.1 \cdot \{0.8 \cdot 0.08 + 0.1 \cdot 0.01 + 0.1 \cdot 0\} = 0.1 \cdot 0.065 = 0.0065$$

Ο ανανεωμένος πίνακας-Q* είναι:

	Αποφάσεις	αριστερά (L)	κάτω (D)	δεξιά (R)	πάνω (U)
Καταστάσεις	[1,1]	0	0	0	0
	[1,2]	0	0	0	0
	[1,3]	0	0	0	0
	[1,4]	0	0	0	0
	[2,1]	0	0	0	0
	[2,2]	0	0	0	0
	[2,3]	0	0	0	0
	[2,4]	0	0	0	0
	[3,1]	0	0	0	0
	[3,2]	0	0.0065	0	0
	[3,3]	0.01	0	0	0
	[3,4]	0	0	0	0
	[4,1]	0	0	0	0
	[4,2]	0	0	0.08	0
	[4,3]	0	0	1	0.0008
	[4,4]	0	0	0	0

Πίνακας 6.5.8: Ανανεωμένος πίνακας Q μετά την προς τα κάτω μετακίνηση του ρομπότ από την κατάσταση [3, 2]*

Το ρομπότ βρίσκεται στην κατάσταση x=[4, 2]. Επιλέγεται τυχαία να μετακινηθεί προς τα αριστερά.

$$Q^*([4,2],L) = R([4,2],L) + 0.1 \cdot \{0.8 \cdot maxQ^*([4,1] + 0.1 \cdot maxQ^*([3,2],u') + 0.1 \cdot maxQ^*([4,2],u')\} = 0 + 0.1 \cdot \{0.8 \cdot 0 + 0.1 \cdot 0.0065 + 0.1 \cdot 0.08\} = 0.1 \cdot 0.00865 = 0.000865$$

Ο ανανεωμένος πίνακας-Q* είναι:

	Αποφάσεις	αριστερά (L)	κάτω (D)	δεξιά (R)	πάνω (U)
Καταστάσεις	[1,1]	0	0	0	0
	[1,2]	0	0	0	0
	[1,3]	0	0	0	0
	[1,4]	0	0	0	0
	[2,1]	0	0	0	0
	[2,2]	0	0	0	0
	[2,3]	0	0	0	0
	[2,4]	0	0	0	0
	[3,1]	0	0	0	0
	[3,2]	0	0.0065	0	0
	[3,3]	0.01	0	0	0
	[3,4]	0	0	0	0
	[4,1]	0	0	0	0
	[4,2]	0.000865	0	0.08	0
	[4,3]	0	0	1	0.0008
	[4,4]	0	0	0	0

Πίνακας 6.5.9: Ανανεωμένος πίνακας Q^* μετά την αριστερή μετακίνηση του ρομπότ από την κατάσταση [4,2]. Τερματισμός επεισοδίου

Στην κατάσταση x=[4, 1] το ρομπότ πέφτει στην τρύπα οπότε το επεισόδιο τερματίζεται.

Ύστερα από 100,000 επαναλήψεις έχουμε τον παρακάτω τελικό πίνακα Q^* όπως αυτός δίνεται από τον κώδικα [62] (Παράρτημα Α)

```
[[3.15720100e-12 7.83083413e-10 5.46108992e-12 6.31996596e-12]
 [9.10220116e-11 2.91800881e-07 1.72152502e-12 1.38242691e-12]
 [1.16948523e-07 1.51014011e-06 7.43361428e-07 4.10775368e-10]
 [6.76587940e-11 6.06023322e-10 1.83643978e-10 1.34143389e-09]
 [6.52175902e-12 1.93643995e-09 2.19261516e-11 3.05721871e-12]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [7.64359046e-07 2.63658463e-11 3.60577799e-08 1.57115719e-10]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [8.13973638e-10 1.21016263e-08 5.38612797e-11 2.38202608e-09]
 [1.63235018e-07 2.32005197e-07 1.77613773e-07 7.41730356e-07]
 [8.36243809e-06 4.70442862e-06 9.35656099e-07 1.24252637e-06]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [4.11604129e-05 1.16766145e-06 1.62181517e-05 5.38015836e-06]
 [1.35384867e-03 1.65942623e-03 3.49578124e-03 7.12364055e-03]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]]
```

Πίνακας 6.5.10: Τελικός πίνακας Q^*

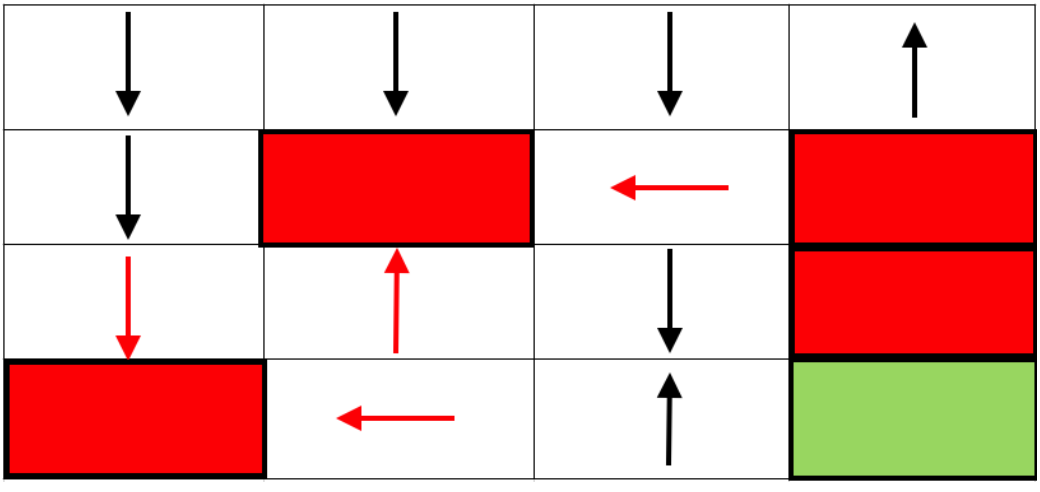
Όπως περιμέναμε ο πίνακας παρουσιάζει μηδενικά στις γραμμές που αντιστοιχούν σε κάποια τελική κατάσταση. Δηλαδή οι γραμμές 6, 8, 12, 13 που αντιστοιχούν στις καταστάσεις [2, 2], [2, 4], [3, 4], [4, 1] είναι μηδενικές λόγω πτώσης του ρομπότ σε τρύπα ενώ η τελευταία κατάσταση [4,4] αντιστοιχεί στην εύρεση του αντικειμένου. Σε κάθε περίπτωση η διαδικασία τερματίζεται. Σε αυτή τη φάση της διαδικασίας εφαρμόζουμε τον τύπο (6.7) για την εύρεση της βέλτιστης πολιτικής. Ισχύει δηλαδή $h^*(x) \in arg\max_u Q^*(x,u)$.

Συνοψίζουμε τα παραπάνω αποτελέσματα σε έναν πιο ευδιάκριτο πίνακα. Κάθε διευρυμένο κελί αποτελείται από 4 μικρότερα κελία τα οποία είναι τοποθετημένα με τέτοιο τρόπο ώστε να αντιστοιχούν σε κάθε δυνατή απόφαση (αριστερά, κάτω, δεξία, πάνω). Σε κάθε υπό-κελί βρίσκονται οι τιμές του τελικού πίνακα Q^* και οι ελάχιστες τιμές είναι αυτές που αντιστοιχούν στη βέλτιστη απόφαση της εκάστοτε κατάστασης και σημειώνονται με κίτρινο χρώμα.

3.157E-12	6.3E-12 7.8E-10	5.46E-12	9.1E-11	1.4E-12 2.9E-07	1.7E-12	1.2E-07	4.1E-10 1.5E-06	7.4E-07	6.8E-11	1.34E-09 6.06E-10	1.8E-10
6.52E-12	3.1E-12 1.9E-09	2.19E-11	0	0 0	0	7.6E-07	1.6E-10 2.6E-11	3.6E-08	0	0 0	0
8.14E-10	2.4E-09 1.2E-08	5.38E-11	1.6E-07	7.4E-07 2.3E-07	1.8E-07	8.4E-06	1.2E-06 4.7E-06	9.4E-07	0	0 0	0
0	0 0	0	4.1E-05	5.4E-06 1.2E-06	1.6E-05	0.00135	0.00712 0.00166	0.00349	0	0 0	0

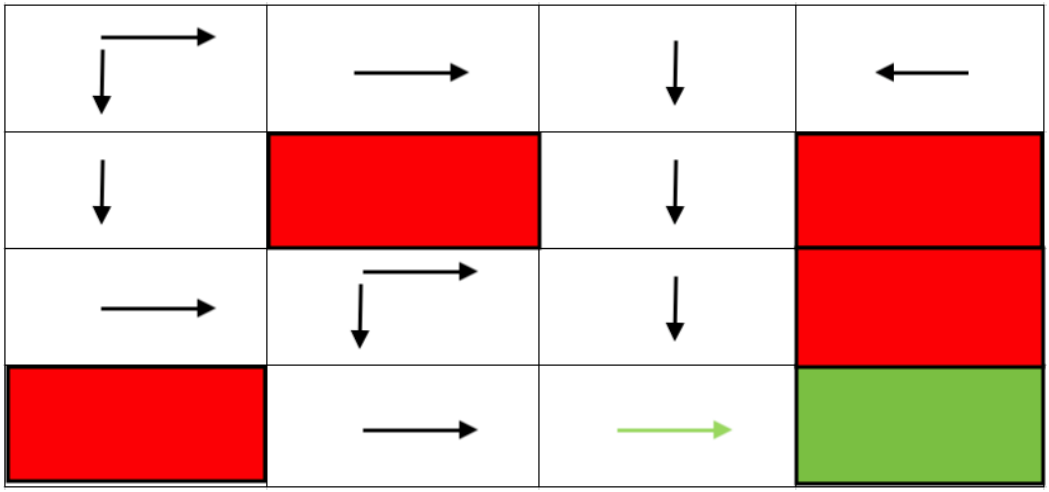
Πίνακας 6.5.11: Τελικός πίνακας Q^*, με κίτρινο χρώμα σημειώνονται οι βέλτιστες αποφάσεις σε κάθε κατάσταση

Όπως έγινε κατανοητό από τα παραπάνω βήματα δεν είναι βέβαιο από ποια κατάσταση θα ξεκινήσει το ρομπότ στο εκάστοτε επεισόδιο. Για αυτό το λόγο άλλωστε, χρειάζεται να βρούμε τη βέλτιστη απόφαση για όλες τις καταστάσεις ώστε να μπορεί να ακολουθεί μία πολιτική από όποια κατάσταση και αν ξεκινήσει. Για παράδειγμα, αν το ρομπότ τύχει να ξεκινήσει από την κατάσταση [1, 1] η βέλτιστη απόφαση είναι να μετακινηθεί προς τα κάτω. Ανάλογα, αν το ρομπότ τύχει να ξεκινήσει από στην κατάσταση [2, 3] η βέλτιστη απόφαση θα είναι η μετακίνηση προς τα αριστερά. Με ανάλογο τρόπο ερμηνεύονται οι αποφάσεις του πίνακα 6.5.10 και συνεπώς ο τελικός πίνακας που παρουσιάζει τη βέλτιστη πολιτική αναλόγως την αρχική κατάσταση του ρομπότ είναι :



Πίνακας 6.5.12: Απεικόνιση βέλτιστης πολιτικής

Παρατηρούμε ότι παρότι μετά από 100Κ επαναλήψεις η διαδικασία συγκλίνει το ρομπότ δεν έχει μάθει πλήρως πως να φτάνει στο στόχο χωρίς να πέφτει σε τρύπα. Σε πολλές καταστάσεις δηλαδή, επιλέγεται ως βέλτιστη απόφαση κάποια που το οδηγεί σε λανθασμένη τερματική κατάσταση. Αυτό μπορεί να οφείλεται σε πολλούς παράγοντες όπως ο συντελεστής γ, ο συντελεστής α που αφορά στο ρυθμό με τον οποίο μαθαίνει το ρομπότ ή ακόμα και στον αριθμό επαναλήψεων. Δηλαδή, ίσως το ρομπότ στη στοχαστική περίπτωση δεν είναι σε θέση να μάθει να φτάνει με επιτυχία στο στόχο σε αυτό τον αριθμό επαναλήψεων. Ενδεικτικά αναφέρουμε ότι το ρομπότ στην ντετερμινιστική περίπτωση φαίνεται να μαθαίνει με επιτυχία μία βέλτιστη διαδρομή και ο πίνακας μιας βέλτιστης πολιτικής δίνεται παρακάτω.



Πίνακας 6.5.13: Απεικόνιση βέλτιστης πολιτικής στη ντετερμινιστική περίπτωση

Όπως φαίνεται σε αυτή την περίπτωση, από όποια κατάσταση και αν ξεκινήσει το ρομπότ μπορεί να ακολουθήσει μία πολιτική που να το οδηγεί στο στόχο. Καταλαβαίνουμε λοιπόν ότι το στοχαστικό πλαίσιο ακόμα και σε πιο απλά παραδείγματα όπως το συγκεκριμένο παιχνίδι μπορεί να είναι πολύ περίπλοκο και χρονοβόρο μέχρι να καταλήξουμε σε ένα επιτυχημένο αποτέλεσμα. Στο συγκεκριμένο στοχαστικό παράδειγμα, το ρομπότ δεν κατάφερε να μάθει μία βέλτιστη διαδρομή στο χρονικό πλαίσιο που του δώσαμε με τα δεδομένα που του ορίσαμε. Όταν το απλοποιήσαμε και ήταν πλέον σαφείς οι κατευθύνσεις των κινήσεων του, μπορούσε δηλαδή να μεταβεί από μία κατάσταση σε μία επόμενη κατάσταση με πιθανότητα 1, κατάφερε στο ίδιο χρονικό πλαίσιο με τον ίδιο ρυθμό μάθησης και προσαρμογής στις αλλαγές του περιβάλλοντος να φτάσει στο αντικείμενο χωρίς να πέσει μέσα σε κάποια τρύπα.

Σε αυτό το σημείο λοιπόν μπορούμε να αναλογιστούμε την επιτυχία των υπολογιστικών προγραμμάτων AlphaGo και AlphaGo Zero που αναφέραμε παραπάνω. Το πρώτο [67] προγραμματίστηκε ώστε να μπορεί να εκπαιδεύεται μέσω ανταμοιβών παίζοντας εκατομμύρια φορές εναντίων ανθρώπων. Έπειτα από αμέτρητα παιχνίδια με επαγγελματίες κατάφερε να κερδίζει τους καλύτερους παίκτες σε ένα παιχνίδι ακόμα πιο περίπλοκο από το σκάκι, δηλαδή σε ένα παιχνίδι που ο αριθμός των πιθανών καταστάσεων και αποφάσεων είναι αναρίθμητος. Το δεύτερο [68] αποτελεί μία εξελιγμένη έκδοση του AlphaGo το οποίο ξεκινώντας από το μηδέν, παίζοντας μόνο με τον εαυτό του κατάφερε μέσα σε λίγες μόνο μέρες να κερδίσει την προηγούμενη έκδοση του AlphaGo 100-0. Σε ένα ευρύτερο πλαίσιο

εφαρμογών καταλαβαίνουμε λοιπόν τις προοπτικές και τη συμβολή της ενισχυτικής μάθησης όπως στη ρομποτική, τη χημεία, τη βιολογία και κάθε άλλο τομέα βοηθώντας την επιστήμη να εξελιχθεί.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Όλα γύρω μας είναι αποφάσεις. Από τις πιο εύκολες μέχρι τις πιο δύσκολες, από αυτές που αφορούν σε ενα άτομο μέχρι αυτές που αφορούν σε έναν ολόκληρο πληθυσμό η λήψη αποφάσεων είναι αναπόσπαστο κομμάτι της ανθρώπινης φύσης και εξέλιξης. Φυσικά το σημαντικό για την επίτευξη αυτής της εξέλιξης είναι η εύρεση βέλτιστων λύσεων κάθε φορά που καλούμαστε να αντιμετωπίσουμε ένα πρόβλημα. Διάφοροι τομείς, όπως οι επιστήμες, η υγεία, η βιομηχανία, το εμπόριο, η οικονομία κ.α., αναζητούν βέλτιστες λύσεις για όλα εκείνα τα περίπλοκα προβλήματα που παρουσιάζονται στην καθημερινότητα. Στην παρούσα εργασία οι εφαρμογές που μελετήσαμε δείχνουν αυτήν ακριβώς την ευρεία εφαρμογή, τη χρησιμότητα και την αποτελεσματικότητα του δυναμικού προγραμματισμού ως μίας σημαντικής μεθόδου βελτιστοποίησης προβλημάτων. Πολλές από τις εφαρμογές στις οποίες υπεισέρχεται, είναι εφαρμογές με τις οποίες ερχόμαστε σε επαφή καθημερινά και ίσως να μην σκεφτόμαστε ότι πίσω από την επίλυσή τους κρύβεται ένας αλγόριθμος δυναμικού προγραμματισμού. Χαρακτηριστικό παράδειγμα αποτελεί η εφαρμογή του τρίτου κεφαλαίου, της εύρεσης απόστασης κατά Levenshtein. Με τους αλγορίθμους που έχουν υλοποιηθεί για την αντιμετώπισή της μπορούμε να μεταφράζουμε εύκολα λέξεις και ολόκληρα κείμενα, να γράφουμε πάντα ορθογραφημένα και να μην ανησυχούμε ότι ο υπολογιστής δεν θα αποκωδικοποιήσει σωστά την αναζήτησή μας σε έναν ιστότοπο λόγω δικής μας ασαφούς σύνταξης του ερωτήματός μας. Ακόμη, όπως είδαμε στο τέταρτο και πέμπτο κεφάλαιο, με την εύρεση ελάχιστων διαδρομών, ο δυναμικός προγραμματισμός έχει συμβάλλει τα μέγιστα όχι μόνο όσον αφορά στις επιχειρήσεις για τη μείωση του κόστους ή του χρόνου μεταφοράς προϊόντων, αλλά και στην καθημερινότητα όλων μας καθώς για τη μετάβαση από μία τοποθεσία σε μία άλλη μπορούμε να βρίσκουμε απευθείας την ελάχιστη διαδρομή ανοίγοντας απλώς την εφαρμογή του Google Maps. Τέλος, ο δυναμικός προγραμματισμός θέτει τα θεμέλια για την εξέλιξη ενός τομέα, της ενισχυτικής μάθησης, ο οποίος καθώς θα αναπτύσσεται θα βρίσκει εφαρμογή σε οτιδήποτε σκεφτούμε, ή σε πράγματα που ούτε έχουμε σκεφτεί ακόμα, από τα αυτόνομα αυτοκίνητα, τα ρομποτικά χέρια που εκτελούν συγκεκριμένες εντολές, μέχρι τα ρομπότ που θα προσομοιώνουν πιστά όχι μόνο τις ανθρώπινες κινήσεις αλλά και τις ανθρώπινες αντιδράσεις και συμπεριφορές.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Hillier, F. S., & Lieberman, G. J. (2001). INTRODUCTION TO OPERATIONS RESEARCH. New York: McGraw-Hill.

[2] ΚΟΛΕΤΣΟΣ, Ι., and Δ. ΣΤΟΓΙΑΝΝΗΣ. "Εισαγωγή 1." ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΕΠΙΧΕΙΡΗΣΙΑΚΗ ΕΡΕΥΝΑ, 3rd ed., Εκδόσεις ΣΥΜΕΩΝ, 2017, pp. 1–14.

[3] Κολέτσος, Ι., & Στογιάννης, Δ. (2021). Επιχειρησιακή Ερευνα. Αθήνα : Εκδόσεις Συμεών.

[4] Ζερβάκης, Σταύρος. "Ο Ρόλος Της Επιχειρησιακής Έρευνας Στη Λήψη Αποφάσεων Και Στον Προγραμματισμό Της Επιχειρηματικής ΔΡαστηριότητας." *ΠΤΥΙΑΚΗ ΕΡΓΑΣΙΑ : Ο Ρόλος Της Επιχειρησιακής Έρευνας Στη Λήψη Αποφάσεων Και Στον Προγραμματισμό Της Επιχειρηματικής ΔΡαστηριότητας, ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΚΡΗΤΗΣ*, 2012, nefeli.lib.teicrete.gr/browse/sdo/log/2012/ZervakisStauros/attached-document-1352363554-955410-25590/ZervakisStauros2012.pdf.

[5] Chinneck, John W. "Chapter 15: Dynamic Programming." *Practical Optimization: a Gentle Introduction*, www.sce.carleton.ca/faculty/chinneck/po.html.

[6] "NP-Completeness: Set 1 (Introduction)." *GeeksforGeeks*, 7 Sept. 2018, www.geeksforgeeks.org/np-completeness-set-1/.

[7] Mullins, Robert. "What Is a Turing Machine?" *Department of Computer Science and Technology – Raspberry Pi: Introduction: What Is a Turing Machine?*, University of Cambridge, 2012, www.cl.cam.ac.uk/projects/raspberrypi/tutorials/turing-machine/one.html.

[8] Landman, Nathan, et al. "P Versus NP." *Brilliant Math & Science Wiki*, 2020, brilliant.org/wiki/p-versus-np/.

[9] Esfahbod, Behnam. "Euler Diagram for P, NP, NP-Complete, and NP-Hard Set of Problems." *Wikipedia*, Wikipedia, 1 Nov. 2007, en.wikipedia.org/wiki/P_versus_NP_problem#/media/File:P_np_np-complete_np-hard.svg.

[10] "PART I: Dynamic Programming." *Dynamic Programming A Computational Tool*, by Art Lew and Holger Mauch, Springer Berlin Heidelberg, 2007, pp. 3–100.

[11] The Levenshtein Distance (Edit Distance) Problem." *Techie Delight*, Techie Delight, 2020, www.techiedelight.com/levenshtein-distance-edit-distance-problem/.

[12] Jurafsky, Dan. "Minimum Edit Distance." *Minimum Edit Distance - Stanford University*, Stanford University, web.stanford.edu/class/cs124/lec/med.pdf.

[13] Langmead, Ben. "Dynamic Programming and Edit Distance." *Dynamic Programming and Edit Distance - JHU Computer ...*, JOHNS HOPKINS WHITING SCHOOL OF ENGINEERING, 2018, www.cs.jhu.edu/~langmea/resources/lecture_notes/dp_and_edit_dist.pdf.

[14] Pavlus, John. “A New Map Traces the Limits of Computation.” *Quanta Magazine*, Quanta Magazine, 29 Sept. 2015, www.quantamagazine.org/edit-distance-reveals-hard-computational-problems-20150929.

[15] Klein, Bernd. “Levenshtein Distance.” *Python Advanced: Recursive and Iterative Implementation of the Edit Distance*, Python Advanced Course Topics, www.python-course.eu/levenshtein_distance.php.

[16] Puntambekar, Renuka. “Cyclic, Acyclic, Sparse & Dense Graphs.” *Computer Science 201: Data Structures & Algorithms* , Study.com, study.com/academy/lesson/cyclic-acyclic-sparse-dense-graphs.html.

[17] Φωτάκης, Δ. “Θεωρία Γραφημάτων: Ορολογία Και Βασικές Έννοιες.” *Graphs_Basics*, Σχολή Ηλεκτρολόγων Μηχανικών Και Μηχανικών Υπολογιστών Εθνικό Μετσόβιο Πολυτεχνείο, 2016, www.softlab.ntua.gr/~fotakis/discrete_math_2015_2016/data/12_Graphs_Basics.pdf.

[18] Chinneck, John W. “Chapter 8: An Introduction to Networks.” *Practical Optimization: a Gentle Introduction*, www.sce.carleton.ca/faculty/chinneck/po.html.

[19] Moore, Karleigh, et al. “Greedy Algorithms.” *Brilliant Math & Science Wiki*, Brilliant.org, brilliant.org/wiki/greedy-algorithm/.

[20] “Εύρεση Απόστασης-Διαδρομής Μεταξύ Πόλεων, Περιοχών Στο Χάρτη. Υπολογισμός Κόστους Ταξιδιού (Καύσιμα, Διόδια).” *Vriskoapostasi.gr*, vriskoapostasi.gr/el/.

[21] Dreyfus, Stuart. *Richard Bellman on the Birth of Dynamic Programming*, Institute for Operations Research and the Management Sciences (INFORMS), 2002, pubsonline.informs.org/doi/pdf/10.1287/opre.50.1.48.17791.

[22] Storchi, Gianni, et al. “9th DIMACS Implementation Challenge - Shortest Paths.” Edited by Umberto Ferraro Petrillo. *DIMACS*, users.diag.uniroma1.it/challenge9/data/rome/rome99.gr?fbclid=IwAR05Si23Av4e4QCkh6JsracETnmYOnBWd_uKbP20_buAP7PwxaQJI69xY-w.

[23] “Dijkstra's Shortest Path Algorithm in Python.” *AlgoTree*, AlgoTree, 2020, algotree.org/algorithms/single_source_shortest_path/dijkstras_shortest_path_python/?fbclid=IwAR1ZakoNZ5kB3XqAMxVJ6Jh-kXy33lQ8IdaEMRS7Eq_O1THqPFHIwJP5F8k.

[24] *Python Dictionaries*, W3Schools, www.w3schools.com/python/python_dictionaries.asp.

[25] *Python Lists*, W3Schools, www.w3schools.com/python/python_lists.asp.

[26] TANIGUCHI, Yasufumi. “Understanding Edge Relaxation for Dijkstra's Algorithm and Bellman-Ford Algorithm.” *Medium*, Towards Data Science, 15 May 2020, towardsdatascience.com/algorithm-shortest-paths-1d8fa3f50769.

[27] Chauhan, Chetan, et al. “Survey of Methods of Solving TSP along with Its Implementation Using Dynamic Programming Approach.” *International Journal of Computer Applications* , vol. 52, 4 Aug. 2012, pp. 12–19., doi:0975 – 8887.

[28] Klitzke, Evan. “The Traveling Salesman Problem Is Not NP-Complete.” *Eklitzke.org*, 9 June 2017, eklitzke.org/the-traveling-salesman-problem-is-not-np-complete.

[29] Balasubramanian, Hari. “The Shortest Path, the Traveling Salesman, and an Unsolved Question.” *3 Quarks Daily*, 3 Quarks Daily, 29 Sept. 2014, 3quarksdaily.com/3quarksdaily/2014/09/the-shortest-path-the-traveling-salesman-and-an-unsolved-question.html.

[30] V. B. Lobo, B. B. Alengadan, S. Siddiqui, A. Minu and N. Ansari, "Traveling Salesman Problem for a Bidirectional Graph Using Dynamic Programming," 2016 International Conference on Micro-Electronics and Telecommunication Engineering (ICMETE), Ghaziabad, 2016, pp. 127-132, doi: 10.1109/ICMETE.2016.26.

[31] Klarreich, Erica. “Computer Scientists Find New Shortcuts for Infamous Traveling Salesman Problem.” *Wired*, Conde Nast, 30 Jan. 2013, www.wired.com/2013/01/traveling-salesman-problem/?fbclid=IwAR02UVmEqOWmiEcB0lRA8-Sk42tJAkgKq9Jw2aL_zWJY9putdpjwnHR1Nzw.

[32]Rodríguez, Alejandro, and Rubén Ruiz. *The Effect of Asymmetry on Traveling Salesman Problems*, 3 Nov. 2010, www.upv.es/deioac/Investigacion/articulo.pdf.

[33] Nguyen, Quang Nhat. *Travelling Salesman Problem and Bellman-Held-Karp Algorithm*, 10 May 2020, www.math.nagoya-u.ac.jp/~richard/teaching/s2020/Quang1.pdf.

[34] “Dynamic Programming and Bit Masking Tutorials & Notes: Algorithms.” *HackerEarth*, www.hackerearth.com/practice/algorithms/dynamic-programming/bit-masking/tutorial/

[35] “Travelling Salesman Problem.” *Wikipedia*, Wikimedia Foundation, 2021, en.wikipedia.org/wiki/Travelling_salesman_problem.

[36] “DAA - Travelling Salesman Problem.” *Tutorialspoint*, www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_travelling_salesman_problem.htm

[37] *Google Maps*, Google, www.google.com/maps/

[38] “Your Maps.” *Build & Collaborate On Beautiful Maps.*, www.atlistmaps.com/maps.

[39] Ekerot, Carl. “Held-Karp.” *GitHub*, 2016, github.com/CarlEkerot/held-karp/blob/master/held-karp.py.

[40] Tripathy, Abhijit. “Travelling Salesman Problem (Bitmasking and Dynamic Programming).” *OpenGenus IQ: Learn Computer Science*, OpenGenus IQ: Learn Computer Science, 5 Sept. 2020, iq.opengenus.org/travelling-salesman-problem-dp/.

[41] “Intractable Problems and DP with Bitmask.” Problem Solving Club. cpc.cpsc.ucalgary.ca/presentations/weekly-meetings/slides/winter2017/2017-03-01-DPBitmask.pdf.

[42] “Http://Www.cs.ucf.edu/~Dmarino/Progcontests/Modules/Dptsp/DP-TSP-Notes.pdf.” *DP TSP NOTES*, 26 Feb. 2015, www.cs.ucf.edu/~dmarino/progcontests/modules/dptsp/DP-TSP-Notes.pdf.

[43] “Bitwise Operators.” *BitwiseOperators - Python Wiki*, Python TM, 2013, wiki.python.org/moin/BitwiseOperators.

[44] “List.” *TSPLIB*, Universität Heidelberg, 2018, comopt.ifi.uni-heidelberg.de/software/TSPLIB95/atsp/

[45] Busoniu, L., Babuska, R., De Schutter, B., & Ernst, D. (2010). Reinforcement Learning and Dynamic Programming Using Function Approximators . CRC Press.

[46] Garychl. “Applications of Reinforcement Learning in Real World.” *Medium*, Towards Data Science, 2 Aug. 2018, towardsdatascience.com/applications-of-reinforcement-learning-in-real-world-1a94955bcd12.

[47] “OpenAI.” *Wikipedia*, Wikimedia Foundation, 2021, en.wikipedia.org/wiki/OpenAI.

[48] Rana, Ashish. “Introduction: Reinforcement Learning with OpenAI Gym.” *Medium*, Towards Data Science, 21 Sept. 2018, towardsdatascience.com/reinforcement-learning-with-openai-d445c2c687d2.

[49] Baker , Monya. “1,500 Scientists Lift the Lid on Reproducibility.” *Nature International Weekly Journal of Science*, Nature Publishing Group, 28 July 2016, www.nature.com/news/1-500-scientists-lift-the-lid-on-reproducibility-1.19970.

[50] OpenAI. “A Toolkit for Developing and Comparing Reinforcement Learning Algorithms.” *Gym*, OpenAI, gym.openai.com/?fbclid=IwAR3mQkS5mvQj6P67h3G0ftu9_rvYQ7G3yTH6ekJEUckfIM66_3Gmq0nNsCA.

[51] Argerich, Mauricio Fadel. “Reinforcement Learning Environments.” *Medium*, Medium, 24 Nov. 2019, medium.com/@mauriciofadelargerich/reinforcement-learning-environments-cff767bc241f.

[52] Lee, Dan. “Reinforcement Learning, Part 3: The Markov Decision Process.” *Medium*, AI³ | Theory, Practice, Business, 30 Oct. 2019, medium.com/ai%C2%B3-theory-practice-business/reinforcement-learning-part-3-the-markov-decision-process-9f5066e073a2.

[53] Lee, Dan. “Reinforcement Learning, Part 4: Optimal Policy Search with MDP.” *Medium*, AI³ | Theory, Practice, Business, 9 Nov. 2019, medium.com/ai%C2%B3-theory-practice-business/reinforcement-learning-part-4-optimal-policy-search-with-mdp-7fc96158ea8a.

[54] TORRES.AI, Jordi. “DRL 02: Formalization of a Reinforcement Learning Problem.” *Medium*, Towards Data Science, 22 May 2020, towardsdatascience.com/drl-02-formalization-of-a-reinforcement-learning-problem-108b52ebfd9a.

[55] TORRES.AI, Jordi. “The Bellman Equation.” *Medium*, Towards Data Science, 11 June 2020, towardsdatascience.com/the-bellman-equation-59258a0d3fa7.

[56] Paul, Sayak. “An Introduction to Q-Learning: Reinforcement Learning.” *FloydHub Blog*, FloydHub Blog, 15 May 2019, blog.floydhub.com/an-introduction-to-q-learning-reinforcement-learning/.

[57] Hanukoglu, Moshe. “Deep Learning and Natural Language Processing.” Deep Reinforcement Learning. www.science.co.il/moshe/documents/deep-learning/Deep_RL/#Deep-Reinforcement-Learning.

[58] OpenAI. “FrozenLake-v0.” *Gym*, OpenAI , gym.openai.com/envs/FrozenLake-v0/.

[59] Sarda, Veena. “Q-Learning Using Python And OpenAI Gym.” *C# Corner*, 3 June 2020, www.c-sharpcorner.com/article/q-learning-using-python-and-openai-gym/?fbclid=IwAR2dAZYtB5n-j4FkIB8amJ__QddGKL6CRhv386XbgMhoyPs68LElopeQArI.

[60] TORRES.AI, Jordi. “DRL 01: A Gentle Introduction to Deep Reinforcement Learning.” *Medium*, Towards Data Science, 15 May 2020, towardsdatascience.com/drl-01-a-gentle-introduction-to-deep-reinforcement-learning-405b79866bf4.

[61] Sunil, Karthik. “ OpenAI Frozen Lake Problem.” *Kaggle*, 2020, www.kaggle.com/karthikcs1/openai-frozen-lake-problem/code?fbclid=IwAR0lBb7CjXJjdXYFQ7NVzcM-YHck4hWOeoIfUCXNZ-6xhGDX_4LCyHu3Z08.

[62] Sunil, Karthik. “OpenAI Frozen Lake Problem.” *Kaggle*, Kaggle, 17 Mar. 2020, www.kaggle.com/karthikcs1/openai-frozen-lake-problem/code?fbclid=IwAR0lBb7CjXJjdXYFQ7NVzcM-YHck4hWOeoIfUCXNZ-6xhGDX_4LCyHu3Z08.

[63] Gass, Saul I., and Arjang A. Assad. “TutORials in Operations Research.” *History of Operations Research*, INFORMS , 3 Sept. 2014, pubsonline.informs.org/doi/pdf/10.1287/educ.1110.0084.

[64] Gaither, Norman. Academy of Management, pp. 71–78, *HISTORICAN DEVELOPMENT OF OPERATIONS RESEARCH*.

[65] “Heuristic (Computer Science).” *Wikipedia*, Wikimedia Foundation, 6 Jan. 2021, en.wikipedia.org/wiki/Heuristic_(computer_science).

[66] “Held–Karp Algorithm.” *Wikipedia*, Wikimedia Foundation, 23 Nov. 2020, en.wikipedia.org/wiki/Held%E2%80%93Karp_algorithm.

[67] D. Silver, A. Huang, A., C.J. Maddison, A. Guez, L. Sifre,G. van den Driessche, J.

Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Has-sabis. Mastering the game of go with deep neuralnetworks and tree search.Nature, 529(7587). 2016.

[68] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen,T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of go without human knowledge.Nature, 2017.

[69] Lee, Dan. "Reinforcement Learning, Part 2: Introducing Markov Process." *Medium*, AI³ | Theory, Practice, Business, 24 Oct. 2019, medium.com/ai%C2%B3-theory-practice-business/reinforcement-learning-part-2-introducing-markov-process-d3586d4003e0.

ΠΑΡΑΡΤΗΜΑ Α

1. Κώδικας εφαρμογής «Απόσταση κατά Levenshtein» [15]

```
import numpy as np


def iterative_levenshtein(str1, str2):
    """
        iterative_levenshtein(s, t) -> ldist
        ldist is the Levenshtein distance between the strings
        s and t.
        For all i and j, dist[i,j] will contain the Levenshtein
        distance between the first i characters of s and the
        first j characters of t
    """

    rows = len(str1)+1
    cols = len(str2)+1
    dist = [[0 for x in range(cols)] for x in range(rows)]

    # source prefixes can be transformed into empty strings
    # by deletions:
    for i in range(1, rows):
        dist[i][0] = i

    # target prefixes can be created from an empty source string
    # by inserting the characters
    for j in range(1, cols):
        dist[0][j] = j

    for col in range(1, cols):
        for row in range(1, rows):
            if str1[row-1] == str2[col-1]:
                cost = 0
            else:
                cost = 1
            dist[row][col] = min(dist[row-1][col] + 1,      # deletion
                                 dist[row][col-1] + 1,      # insertion
                                 dist[row-1][col-1] + cost) # substitution

    # for r in range(rows):
    #     print(dist[r])


    return dist[row][col]



filename = open('ancient_greek_names.txt', 'r', encoding='utf-8')
lines = filename.readlines()
num_of_lines = len(lines)
# print(num_of_lines)

str1="nepheli"
str1=str1.lower()

D = {}
T={}

for i in range(0, num_of_lines):
    line = lines[i]
    line = line.rstrip("\n")
```

```
    values = line.split("\t")
    str2 = values[0]
    str2=str2.lower()

    translation=values[1]

    dist=iterative_levenshtein(str1,str2)

    D[str2] = dist
    T[translation]= dist


    # print(dist,str2)
    # print(iterative_levenshtein(str1,str2))

S = sorted(D.items(), key=lambda x: x[1])
A = sorted(T.items(), key=lambda x: x[1])
print(S)
print()
print(A)
```

Agathippi Αγαθίππη	Delfis Δελφίς	Iokasti Ιοκάστη
Amaryllis Αμαρυλλίς	Diianeira Δηιάνειρα	Ippolyti Ιππολύτη
Ariadni Αριάδνη	Diodwra Διοδώρα	Iris Ίρις
Armonia Αρμονία	Diidameia Δηϊδάμεια	Ifigeneia Ιφιγένεια
Artemis Άρτεμις	Diwni Διώνη	Ifinoi Ιφινόη
Aithra Αίθρα	Didw Διδώ	Kallirroi Καλλιρρόη
Aktis Ακτίς	Dwrothea Δωροθέα	Kalypsw Καλυψώ
Alkistis Άλκηστις	Ekali Εκάλη	Kleiw Κλειώ
Alkmini Αλκμήνη	Erifyli Εριφύλη	Kleopatra Κλεοπάτρα
Amaltheia Αμάλθεια	Eris Έρις	Klewni Κλεώνη
Afroditi Αφροδίτη	Eratw Ερατώ	Korwnis Κορωνίς
Arsinoi Αρσινόη	Ermioni Ερμιόνη	Litw Λητώ
Agisistrati Αγησιστράτη	Ilektra Ηλέκτρα	Leia Λεία
Anthemis Άνθεμις	Irofili Ηροφίλη	Melpomeni Μελπομένη
Aoidi Αοιδή	Irigoni Ηριγόνη	Melitta Μέλιττα
Avgi Αυγή	Irw Ηρώ	Myrti Μύρτη
Vrisiida Βρισηίδα	THaleia Θάλεια	Maira Μαίρα
Galateia Γαλάτεια	THemis Θέμις	Maris Μάρις
Gaia Γαία	THetis Θέτις	Myrtw Μυρτώ
Giasemi Γιασεμ	Ianthi Ιάνθη	Myrtia Μυρτιά

Mnisareti Μνησαρέτη	Agisilaos Αγησίλαος	Defkaliwn Δευκαλίων
Niis Νηίς	Adeimantos Αδείμαντος	Evrysthenis Ευρυσθένης
Nafsika Ναυσικά	Aigisthos Αίγισθος	Zeyksippos Ζεύξιππος
Neaira Νέαιρα	Aiolos Αίολος	Igisiklis Ηγησικλής
Nefeli Νεφέλη	Aineias Αινείας	Irakleitos Ηράκλειτος
Niriida Νηρηίδα	Aisxinis Αισχίνης	THalis Θαλής
KSanthippi Ξανθίππη	Aisxylos Αισχύλος	THoukydidis Θουκυδίδης
Oianthi Οιάνθη	Alkiviadis Αλκιβιάδης	Ifiklis Ιφικλής
Pinelopi Πηνελόπη	Ameinias Αμεινίας	Iaswn Ιάσων
Persiida Περσηίδα	Anaksandridis Αναξανδρίδης	Iwn Ίων
Polykseni Πολυξένη	Anaksagoras Αναξαγόρας	Ikaros Ίκαρος
Rwksani Ρωξάνη	Anaksimenis Αναξιμένης	Iwsipos Ιώσηπος
Terpsixori Τερψιχόρη	Andokidis Ανδοκίδης	Kimwn Κίμων
Faidra Φαίδρα	Aratos Άρατος	Kleanthis Κλεάνθης
Fereniki Φερενίκη	Arrianos Αρριανός	Kleomenis Κλεομένης
Foivi Φοίβη	Daidalos Δαίδαλος	Kritwn Κρίτων
Xrysiis Χρυσηίς	Diifovos Δηίφοβος	Kallistratos Καλλίστρατος
Xioni Χιόνη	Diaios Δίαιος	Leandros Λέανδρος
Xloi Χλόη	Diodotos Διόδοτος	Lewn Λέων
	Diogenis Διογένης	Lewnidas Λεωνίδας

Linos Λίνος	Polydeykis Πολυδεύκης	
Lykoyrgos Λυκούργος	Pyrros Πύρρος	
Lysandros Λύσανδρος	Polydwros Πολύδωρος	
Maiwnas Μαίωνας	Solwn Σόλων	
Menelaos Μενέλαος	Sofoklis Σοφοκλής	
Menaixmos Μέναιχμος	Swkratis Σωκράτης	
Miltiadis Μιλτιάδης	Tilemaxos Τηλέμαχος	

Neoklis	Νεοκλής	Timolewn	Τιμολέων	
Nireas	Νηρέας	Yakinthos	Υάκινθος	
Nikainetos	Νικαίνετος	Faidwn	Φαίδων	
KSenofwn	Ξενοφών	Filoktitis	Φιλοκτήτης	
KSenagoras	Ξεναγόρας	Filoymenos	Φιλούμενος	
Odysseas	Οδυσσέας	Filyllios	Φιλύλλιος	
Oneiros	Όνειρος	Filwn	Φίλων	
Orestis	Ορέστης	Foivos	Φοίβος	
Paris	Πάρις	Filagoras	Φιλαγόρας	
Patroklos	Πάτροκλος	Friksos	Φρίξος	
Pythagoras	Πυθαγόρας	Frontis	Φρόντις	
Prwtagoras	Πρωταγόρας	Wtos	Ώτος	
Periklis	Περικλής			

2. Κώδικας εφαρμογής Ελάχιστης Διαδρομής (Shortest Path Problem) [23]

```
1.  class Node_Distance :
2.
3.      def __init__(self, name, dist) :
4.          self.name = name
5.          self.dist = dist
6.
7.  class Graph :
8.
9.      def __init__(self, node_count) :
10.         self.adjlist = {}
11.         self.node_count = node_count
12.
13.     def Add_Into_Adjlist(self, src, node_dist) :
14.         if src not in self.adjlist :
15.             self.adjlist[src] = []
16.         self.adjlist[src].append(node_dist)
17.
18.     def Dijkstras_Shortest_Path(self, source) :
19.
20.         f = open("path.txt", "w")
21.
22.         # Initialize the distance of all the nodes from source to infinity
23.         distance = [999999999999] * self.node_count
24.         # Distance of source node to itself is 0
25.         distance[source] = 0
26.
27.         path = []
28.         for i in range(3354):
29.             path.append("")
30.
31.
32.         # Create a dictionary of { node, distance_from_source }
33.         dict_node_length = {source: 1}
34.
35.
36.         while dict_node_length :
37.
38.             # Get the key for the smallest value in the dictionary
39.             # i.e Get the node with the shortest distance from the source
40.             source_node = min(dict_node_length, key = lambda k: dict_node_len
    gth[k])
41.             del dict_node_length[source_node]
42.
43.
44.             for node_dist in self.adjlist[source_node] :
```

```
45.                 adjnode = node_dist.name
46.                 length_to_adjnode = node_dist.dist
47.
48.                 # Edge relaxation
49.                 if distance[adjnode] > distance[source_node] + length_to_adjn
    ode :
50.                     distance[adjnode] = distance[source_node] + length_to_adj
    node
51.                     dict_node_length[adjnode] = distance[adjnode]
52.
53.                     path[adjnode] = path[source_node] + str(adjnode) + " "
54.
55.
56.         for i in range(self.node_count) :
57.             #f.write("Source Node ("+str(source)+")  -
    > Destination Node(" + str(i) + ")  : " + str(distance[i]) + "\n")
58.             f.write("Source Node ("+str(source)+")  -
    > Destination Node(" + str(i) + ")  : " + str(distance[i]) + " " + str(path[i
    ]) + "\n")
59.
60.         f.close()
61.
62.
63. def main() :
64.
65.     g = Graph(3354)
66.
67.
68.     infile = open("rome_graph_full.txt","r")
69.     for line in infile:
70.         line = line.rstrip("\n")
71.         elements = line.split('\t')
72.         node = int(elements[0])
73.         destination = int(elements[1])
74.         cost = int(elements[2])
75.         #print(node)
76.         #print(destination)
77.         #print(cost)
78.         #print(node + "..." + destination + "..." + cost)
79.         g.Add_Into_Adjlist(node, Node_Distance(destination, cost))
80.
81. #     g.Dijkstras_Shortest_Path(0)
82.     print("\n")
83.     g.Dijkstras_Shortest_Path(1)
84. #     g.Dijkstras_Shortest_Path(3353)
85.
86.
87. if __name__ == "__main__" :
88.       main()
```

Τα δεδομένα για την εφαρμογή του προβλήματος ελάχιστης διαδρομής βρέθηκαν από το κέντρο DIMACS για διακριτά μαθηματικά και θεωρητική επιστήμη των υπολογιστών (Storchi, Gianni, et al. "9th DIMACS Implementation Challenge - Shortest Paths." Edited by Umberto Ferraro Petrillo. *DIMACS*, users.diag.uniroma1.it/challenge9/data/rome/rome99.gr?fbclid=IwAR05Si23Av4e4QCkh6JsracETnmYOnBWd_uKbP20_buAP7PwxaQJI69xY-w)

3. Κώδικας εφαρμογής Πλανόδιου Πωλητή (Travelling Salesman Problem) [39]

```
import itertools
import random
import sys



def held_karp(dists):
    """
    Implementation of Held-Karp, an algorithm that solves the Traveling
    Salesman Problem using dynamic programming with memoization.
    Parameters:
        dists: distance matrix
    Returns:
        A tuple, (cost, path).
    """
    n = len(dists)

    # Maps each subset of the nodes to the cost to reach that subset, as well

    # as what node it passed before reaching this subset.
    # Node subsets are represented as set bits.
    C = {}

    # Set transition cost from initial state
    for k in range(1, n):
        C[(1 << k, k)] = (dists[0][k], 0)


    # Iterate subsets of increasing length and store intermediate results
    # in classic dynamic programming manner
    for subset_size in range(2, n):
        for subset in itertools.combinations(range(1, n), subset_size):

            # Set bits for all nodes in this subset
            bits = 0
            for bit in subset:
               bits |= 1 << bit

            # Find the lowest cost to get to this subset
            for k in subset:
                prev = bits & ~(1 << k)


                res = []
                for m in subset:
                    if m == 0 or m == k:
                        continue
                    res.append((C[(prev, m)][0] + dists[m][k], m))

                C[(bits, k)] = min(res)


    # We're interested in all bits but the least significant (the start state
)
    bits = (2**n - 1) - 1


    # Calculate optimal cost
    res = []
```

```
    for k in range(1, n):
        res.append((C[(bits, k)][0] + dists[k][0], k))
    opt, parent = min(res)

    # Backtrack to find full path
    path = []
    for i in range(n - 1):
        path.append(parent)
        new_bits = bits & ~(1 << parent)
        _, parent = C[(bits, parent)]


        bits = new_bits


    # Add implicit start state
    path.append(0)


    return opt, list(reversed(path))


def generate_distances(n):
    dists = [[0] * n for i in range(n)]
    for i in range(n):
        for j in range(i+1, n):
            dists[i][j] = dists[j][i] = random.randint(1, 99)
            print(dists)

    return dists


def read_distances(filename):


    hk = open(filename,'r')
    lines = hk.readlines()
    num_of_lines = len(lines)
    dists = [[0] * num_of_lines for i in range(num_of_lines)]

    for i in range(num_of_lines):

        line = lines[i]
        if line[0] == '#':
            continue
        line=line.rstrip('\n')
        elements=line.split(',')
        num_of_values = len(elements)


        for j in range(num_of_values):

            dists[i][j] = int(elements[j])


    return dists


if __name__ == '__main__':
    arg = sys.argv[1]

    if arg.endswith('.txt'):
        dists = read_distances(arg)
    else:
        dists = generate_distances(int(arg))

```

```
    # Pretty-print the distance matrix
    # for row in dists:
    #     print(''.join([str(n).rjust(3, ' ') for n in row]))

    print('')

    print(held_karp(dists))
```

Δεδομένα για πρόβλημα Πλανόδιου Πωλητή 17 κόμβων (“List.” *TSPLIB*, Universität Heidelberg, 2018, comopt.ifi.uni-heidelberg.de/software/TSPLIB95/atsp/)

```
9999, 3,    5,    48,   48,    8,    8,    5,    5,    3,    3,    0,
3,    5,     8,    8,    5
3,    9999, 3,    48,   48,    8,    8,    5,    5,    0,    0,    3,
0,    3,     8,    8,    5
5,    3,    9999, 72,   72,    48,   48,   24,   24,   3,    3,    5,
3,    0,     48,   48,   24
48,   48,   74,   9999, 0,     6,    6,    12,   12,   48,   48,   48,
48,   74,    6,    6,    12
48,   48,   74,   0,    9999,  6,    6,    12,   12,   48,   48,   48,
48,   74,    6,    6,    12
8,    8,    50,   6,    6,     9999, 0,    8,    8,    8,    8,    8,
8,    50,    0,    0,    8
8,    8,    50,   6,    6,     0,    9999, 8,    8,    8,    8,    8,
8,    50,    0,    0,    8
5,    5,    26,   12,   12,    8,    8,    9999, 0,    5,    5,    5,
5,    26,    8,    8,    0
5,    5,    26,   12,   12,    8,    8,    0,    9999, 5,    5,    5,
5,    26,    8,    8,    0
3,    0,    3,    48,   48,    8,    8,    5,    5,    9999, 0,    3,
0,    3,     8,    8,    5
3,    0,    3,    48,   48,    8,    8,    5,    5,    0,    9999, 3,
0,    3,     8,    8,    5
0,    3,    5,    48,   48,    8,    8,    5,    5,    3,    3,
9999, 3,    5,     8,    8,    5
3,    0,    3,    48,   48,    8,    8,    5,    5,    0,    0,    3,
9999, 3,     8,    8,    5
5,    3,    0,    72,   72,    48,   48,   24,   24,   3,    3,    5,
3,    9999,  48,   48,   24
8,    8,    50,   6,    6,     0,    0,    8,    8,    8,    8,    8,
8,    50,    9999, 0,    8
8,    8,    50,   6,    6,     0,    0,    8,    8,    8,    8,    8,
8,    50,    0,    9999, 8
5,    5,    26,   12,   12,    8,    8,    0,    0,    5,    5,    5,
5,    26,    8,    8,    9999
```

4. Κώδικας εφαρμογής Παγωμένης Λίμνης [62].

```
import gym
import time
from IPython.display import clear_output

env = gym.make("FrozenLake-v0", is_slippery=True).env
# env.s = 10
env.render()

# for i in range(0,100):
#     clear_output(wait=True)
#     env.reset()
#     env.render()
#     time.sleep(0.5)

def print_frames(frames):
    for i, frame in enumerate(frames):
        clear_output(wait=True)
        print(frame['frame'])
        print(f"Episode: {frame['episode']}")
        print(f"Timestep: {i + 1}")
        print(f"State: {frame['state']}")
        print(f"Action: {frame['action']}")
        print(f"Reward: {frame['reward']}")
        time.sleep(1)

import numpy as np
q_table = np.zeros([env.observation_space.n, env.action_space.n])

state = env.reset()
env.s = 14
env.render()
print(env.step(2))
time.sleep(5)
clear_output(wait=True)
env.render()

%%time
"""Training the agent"""

import random
from IPython.display import clear_output

# Hyperparameters
alpha = 0.8
gamma = 0.1
epsilon = 0.2

# For plotting metrics
all_epochs = []
# all_penalties = []

for i in range(1, 100000):
    state = env.reset()

    epochs,  reward, = 0, 0
    done = False

    while not done:
        explore_eploit = random.uniform(0, 1)
        if  explore_eploit < epsilon:
            action = env.action_space.sample() # Explore action space
```

```
        else:
            action = np.argmax(q_table[state]) # Exploit learned values

        next_state, reward, done, info = env.step(action)

        old_value = q_table[state, action]
        next_max = np.max(q_table[next_state])

        new_value = (1 - alpha) * old_value + alpha * (reward + gamma * next_
max)
        q_table[state, action] = new_value

        state = next_state
        epochs += 1

    if i % 100 == 0:
        clear_output(wait=True)
        print(f"Episode: {i}")

print("Training finished.\n")
total_epochs = 0
episodes = 100
frames = []
tot_reward = 0
from random import randint

for ep in range(episodes):
#     state = env.reset()
    env.s = randint(0, 13)
    epochs, reward =  0, 0

    done = False

    while not done:
        action = np.argmax(q_table[state])
        state, reward, done, info = env.step(action)


        # Put each rendered frame into dict for animation
        frames.append({
            'frame': env.render(mode='ansi'),
            'episode': ep,
            'state': state,
            'action': action,
            'reward': reward
            }
        )
        epochs += 1

    total_epochs += epochs
    tot_reward += reward

print(f"Results after {episodes} episodes:")
print(f"Average timesteps per episode: {total_epochs / episodes}")
print(f"Total Rewards {tot_reward}")
print_frames(frames)
q_table
```