ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ο ΑΛΓΟΡΙΘΜΟΣ ΤΗΣ ΑΠΟΙΚΙΑΣ ΜΥΡΜΗΓΚΙΩΝ ΚΑΙ ΕΦΑΡΜΟΓΕΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Μπίκας Ευάγγελος

ΕΠΙΒΛΕΠΩΝ: Δραζιωτης Κωνσταντινος

Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Σχολή Θετικών Επιστημών
Τμήμα Πληροφορικής

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ρητά ότι η παρούσα πτυχιακή εργασία, καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στο πλαίσιο αυτής της εργασίας, αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή/και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής.

Το περιεχόμενο αυτής της εργασίας δεν απηχεί απαραίτητα τις απόψεις του Τμήματος, του Επιβλέποντα, ή της επιτροπής που την ενέκρινε.

Υπεύθυνη Δήλωση

(Υπογραφή) ....

Μπίκας Ευάγγελος

Περίληψη

Σκοπός της παρούσας πτυχιακής εργασίας είναι η παρουσίαση και υλοποίηση του αλγόριθμου της αποικίας των μυρμηγκιών. Ο αλγόριθμος αυτός είναι ένας αλγόριθμος βελτιστοποίησης που είναι βασισμένος σε παρατηρήσεις που έγιναν στα μυρμήγκια στην φύση. Πιο συγκεκριμένα, έχει μελετηθεί ο τρόπος με τον οποίο καταφέρνουν να αλληλεπιδρούν μεταξύ τους και το πως βρίσκουν πάντα την βέλτιστη διαδρομή από την φωλιά τους μέχρι την πηγή τροφής.

Αυτό το πετυχαίνουν εκπέμπτοντας χημικές ουσίες που τις χρησιμοποιούν ως σήματα επικοινωνίας, τις λεγόμενες φερομόνες. Ο αλγόριθμος που θα αναλύσουμε είναι μία προσομοίωση αυτής της συμπεριφοράς με χρήση τεχνητών μυρμηγκιών.

Το γεγονός ότι βρίσκει βέλτιστη λύση σε συνδυασμό με το πόσο γρήγορα επιτυγχάνεται και το ότι μπορούν να εφαρμοστούν πολλές παραλλαγές του ανάλογα με το ζητούμενο καθιστά αυτόν τον αλγόριθμο χρήσιμο σε διάφορους τομείς, όπως η επεξεργασία εικόνων και οι τηλεπικοινωνίες. Εντάσεται σε πολλά πεδία και εφαρμογές που έχουν ως στόχο την βελτιστοποίηση, με κλασικό παράδειγμα αυτό του πλανόδιου πωλητή.

Παρακάτω, θα αναλυθεί η θεωρία γράφων και το πως σχετίζεται με αλγόριθμους βελτιστοποίησης, θα αναφερθούν κάποιοι αλγόριθμοι βελτιστοποίησης με έμφαση στον αλγόριθμο της αποικίας των μυρμηγκιών. Θα παρουσιαστεί η δική μου υλοποίηση, με διάφορα παραδείγματα εκτέλεσης του και θα δωθούν πιθανές εφαρμογές σε προβλήματα πραγματικού κόσμου σε τομείς όπως η επιστήμη των υπολογιστών. Όλοι οι αλγόριθμοι θα υλοποιηθούν σε python.

Λέξεις Κλειδιά. Προβλήματα Βελτιστοποίησης, Αλγόριθμος Αποικίας Μυρμηγκιών, Θεωρία Γράγων , Python

SUMMARY

The purpose of this thesis is the presentation and implementation of the ant colony algorithm. This algorithm is an optimization algorithm that is based on observations made on ants in nature. More specifically, the way in which they manage to interact with each other and how they always find the optimal route from their nest to the food source has been studied.

They achieve this by emitting chemicals that they use as communication signals, so-called pheromones. The algorithm we will analyze is a simulation of this behavior using artificial ants.

The fact that it finds an optimal solution combined with how fast it is achieved and that many variations of it can be applied depending on the application make this algorithm useful in various fields such as image processing and telecommunications, and span many fields and applications. which aim at optimization, with the classic example of that of the traveling salesman.

Below we will discuss graph theory and how it relates to optimization algorithms, introduce some optimization algorithms with an emphasis on the ant colony algorithm, present my own implementation and compare it to variations of this and other optimization algorithms, and give possible applications in real-world problems in fields such as computer science. All algorithms will be implemented in python.

Key Words. Optimization Problems, Ant Colony Algorithm, Graph Theory, Python

Περιεχόμενα

ΚΕΦΑΛΑΙΟ 1

Εισαγωγή

Ένα μεμωνομένο μυρμήγκι δεν μπορεί να κάνει πολλά, τα μυρμήγκια ως σύνολο όμως, όπως και κάθε άλλο είδος εντόμων που λειτουργούν ομαδικά ως σμήνος για την επίτευξη κοινού στόχου, μπορούν να ολοκληρώσουν εξιδεικευμένες εργασίες γρήγορα και αποτελεσματικά. Αυτό αποτελεί πηγή έμπνευσης για τον αλγόριθμο που θα μελετήσουμε σε αυτήν την εργασία καθώς και για πολλούς άλλους αλγόριθμους σμήνους που έχουν ως στόχο την επίλυση πολύπλοκων προβλημάτων βελτιστοποίησης πραγματικού κόσμου. [6]

Τα τελευταία χρόνια, τέτοια προβλήματα παρουσιάζονται σε διάφορους κλάδους με πληθώρα θεωρητικών και πρακτικών εφαρμογών όπως το πρόβλημα γεφυρών και το πρόβλημα του πλανόδιου πωλητή (Traveling Salesman Problem - TSP) καθώς και σε κοινωνικά δίκτυα, δρομολόγηση, προβλήματα προσβασιμότητας, βελτιστοποίηση δικτύων επικοινωνίας, σχεδίαση ηλεκτρικών κυκλωμάτων, κ.λπ., για το λόγο αυτό, ιδιαίτερο ενδιαφέρον έχει αναπτυχθεί για την μελέτη τους.

Ο αλγόριθμος της αποικίας μυρμηγκιών (Ant Colony Optimization - ACO) που θα αναλυθεί στην παρούσα πτυχιακή εργασία είναι ένας αλγόριθμος νοημοσύνης σμήνους και με χρήση αυτού αντιμετωπίζονται τέτοια προβλήματα. Ο όρος "Νοημοσύνη Σμήνους" πρωτοεμφανίστηκε το 1989 από τους Gerardo Beni και Jing Wang και μας βοηθάνε στην επίλυση προβλημάτων βελτιστοποίσης, ενώ το 1992 προτάθηκε ο ACO μέσω της διδακτορικής διατριβής του Marco Dorigo. Οι αλγόριθμοι αυτοί μιμούνται εξελικτικές διαδικασίες που παρουσιάζονται στην φύση όπως η επιλογή μιας διαδρομής και η προσαρμογή σε νέα δεδομένα. Ο αλγόριθμος της απεικείας των μυρμηγκιών συγκεκριμένα βασίζεται, όπως προκείπτει κι από το όνομα του, στη συμπεριφορά των μυρμηγκιών στην φύση και το πώς καταφέρνουν πάντα να βρίσκουν την βέλτιστη διαδομή σε όποιο πρόβλημα τους παρουσιαστεί με χρήση πιθανολογικών τεχνικών. Πέρα από τον ACO, παραδείγματα άλλων τέτοιων αλγορίθμων είναι ο αλγόριθμος βελτιστοποίησης σμήνους σωματιδίων (particle swarm optimization - PSO), ο αλγόριθμος των μελισσών (Bee Colony Algorithm - BCA), οι διάφοροι γενετικοί αλγόριθμοι (Genetic Algorithms- GA) και πολλοί άλλοι.

Ένα κεφάλαιο απαραίτητο για την υλοποίηση αυτού του αλγόριθμου είναι η θεωρία γραφών. Έννοιες στις οποίες βασίζεται ο ACO όπως η φερομόνης και η ευρετική συνάρτηση μοντελοποιούνται με χρήση αυτής. Κάθε ακμή του γράφου αντιστοιχεί σε μία τιμή φερομόνης, η οποία ανανεώνεται καθώς τα μυρμήγκια περνούν από αυτήν. Τα μυρμήγκια εξερευνούν πιθανές διαδρομές, και ενισχύουν την ποσότητα φερομόνης στις καλύτερες, επηρεάζοντας έτσι τις επιλογές των μυρμηγκιών που ακολουθούν δίνοντας στον αλγόριθμο την δυνατότητα να "θυμάται" προηγούμενες λύσεις και να τις αξιοποιεί στο μέλλον.

I am lost! Where is the line?!
—A Bug's Life, Walt Disney, 1998[6]

ΚΕΦΑΛΑΙΟ 2

Θεωρία Γράφων

2.1 Ιστορική Αναδρομή

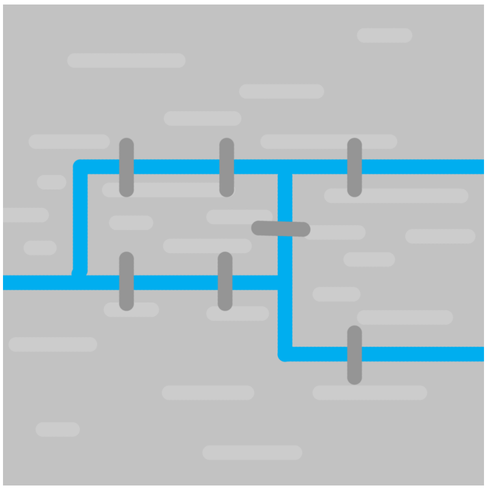

Figure 1: Προσομοίωση Κόνιγκσμπεργκ.

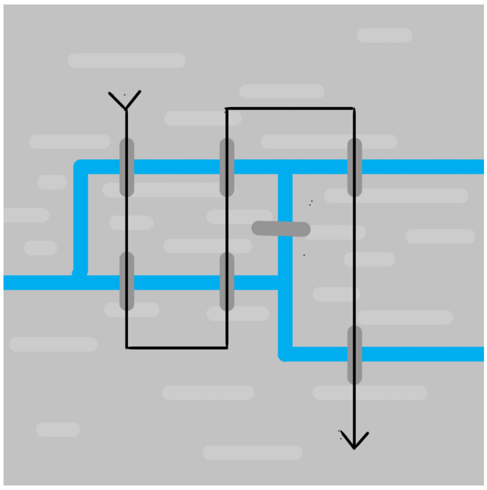

Figure 2: Παράδειγμα διάσχισης γεφυρών

Η ανάπτυξη της θεωρίας γράφων ξεκίνησε τον 18ο αιώνα και πιο συγκεκριμένα το 1736 στην πόλη Königsberg της Πρωσσίας. Σήμερα είναι το Ρωσικό Kaliningrad (μεταξύ Λιθουανίας και Πολωνίας στη Βαλτική) [15]. Η πόλη ήταν χωρισμένη σε 4 τμήματα από τον ποταμό Pregel και χρησιμοποιούνταν 7 γέφυρες για να γίνεται εφικτή η διέλευση των κατοίκων στα διάφορα τμήματά της [Figure 1]. Όταν ο Ελβετός μαθηματικός Λέονχαρντ Όυλερ αναρωτήθηκε αν είναι εφικτό να διασχίσει κάποιος τις γέφυρες της πόλης με βασικό περιορισμό να διασχιστούν όλες οι γέφυρες μόνο μία φορά (Πρόβλημα των γεφυρών του Κόνιγκσμπεργκ)[Figure 2], έτσι ένας νέος κλάδος των διακριτών μαθηματικών γεννήθηκε, γνωστός και ως θεωρία γράφων. Ο Όυλερ απέδειξε ότι δεν υπάρχει τέτοια διαδρομή μέσω

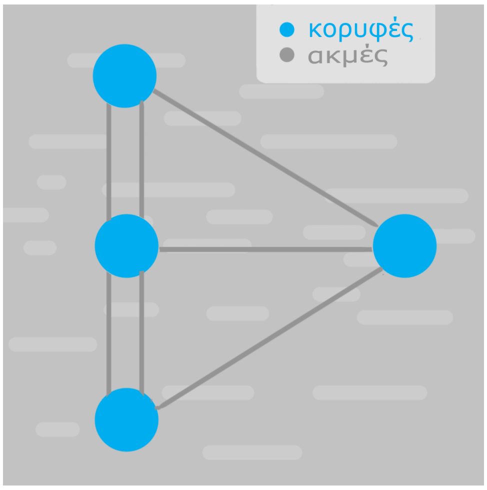

Figure 3: Κόνιγκσμπεργκ ως γράφος.

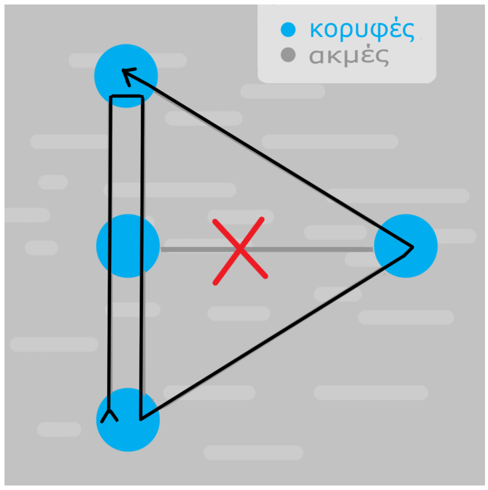

Figure 4: Διαδρομή Όυλερ.

της χρήσης γράφων και κατά συνέπεια το πρόβλημα δεν έχει λύση . Αυτή η απόδειξη απέκτισε αξία όταν ο Ούλερ την εφάρμοσε και σε άλλα προβλήματα γράφων και γενίκευσε την βασική ιδέα.

Μια διαδρομή ονομάζεται διαδρομή Ούλερ όταν μπορούμε να επισκεφτούμε κάθε περιοχή-κορυφή διασχίζοντας την κάθε γέφυρα-ακμή μόνο μία φορά (και ονομάζεται κυκλική αν καταλλήγουμε εκεί που ξεκινήσαμε), αν υπάρχει μια τέτοια διαδρομή σε ένα γράφο τότε αυτός ο γράφος ονομάζεται γράφος Όυλερ. [16] Στο σχήμα [Figure 3] που αντιπροσοπεύει την πόλη του Κόνιγκσμπεργκ σε μορφή γράφου δεν υπάρχει μια τέτοια διαδρομή. Για να γίνει αυτό πρέπει να αφαιρέσουμε μία γέφυρα-ακμή [Figure 4]. Παρατηρήθηκε από τον Όυλερ ότι όλες οι κορυφές πρέπει να έχουν άρτιο βαθμό εκτός από αυτές που ξεκινά και τελειώνει η διαδρομή, εκτός κι αν η διαδρομή είναι κυκλική.

2.2 Εισαγωγή στην Θεωρία Γράφων

Στην ουσία, ένας γράφος είναι διάσπαρτα σημεία (κορυφές) που ενώνονται με γραμμές (ακμές). Γράφους μπορούμε να συνατήσουμε σε διάφορα προβλήματα της καθημερινότητας όπως δίκτυα υποδομών (πχ δίκτυο ύδρευσης), προβλήματα χαρτογράφησης (πλοήγηση), τηλεπικοινωνιών (πχ δορυφόροι), μεταφορών (πχ σιδηρόδρομοι) , και άλλα [15]. Η θεωρία γράφων είναι ένας σημαντικός τομέας των μαθηματικών γιατί πέρα απ' το γεγονός ότι με χρήση αυτών μπορούμε να μοντελοποιήσουμε εύκολα προβλήματα της καθημερινότητας μας σε τομείς που αναφέρθηκαν παραπάνω, μπορούμε επίσης να αναπτύξουμε αλγόριθμους που λύνουν προβλήματα με χρήση γράφων. Παράδειγμα αυτού είναι και ο αλγόριθμος αποικίας των μυρμηγκιών που θα αναλύσουμε σε αυτήν την πτυχιακή εργασία.

2.3 Βασικοί Ορισμοί και έννοιες

Για να γίνουν κατανοητά όσα θα αναφερθούν στην παρακάτω εργασία είναι απαραίτητο να παρουσιαστεί το θεωρητικό υπόβαθρο πάνω στο οποίο είναι βασισμένοι οι αλγόρθμοι βελτιστοποίησης. Για πιο αναλυτική μελέτη παραπέμπονται οι βιβλιογραφικές αναφορές που χρησιμοποιήθηκαν στο τέλος της εργασίας.

Ένας γράφος είναι μία μαθηματική δομή που ορίζεται με αυστηρό τρόπο μέσω δύο συνόλων: το σύνολο κόμβων (ή κορυφών) και το σύνολο ακμών (ή γραμμών) που συνδέουν ζεύγη κορυφών μεταξύ τους και χρησιμοποιείται για την αναπαράσταση πληροφορίας σχετικά με συνδεσμολογία [16]. Όταν δύο κορυφές είναι συνδεδεμένες, δηλαδή ενώνονται με μία ακμή ονομάζονται γειτονικές (για παραδειγμα στο [Figure 5] η κορυφή "A" και η κορυφή "B" είναι γειτονικές), αντίστοιχα δύο ακμές που καταλήγουν σε ίδια κορυφή ονομάζονται προσπίπτουσες της κορυφής αυτής. Το πόσες ακμές προσπίπτουν σε μία κορυφή είναι ο βαθμός της κορυφής αυτής. Τάξη ενός γράφου καλούμε το πόσες κορυφές έχει (για παράδειγμα στο [Figure 5] η κορυφή "E" είναι τέταρτου βαθμού και ο γράφος είναι έκτης τάξης) . Ένας γράφος μπορεί να είναι είτε κατευθυνόμενος όταν οι ακμές έχουν μια κατεύθυνση από έναν κόμβο προς έναν άλλο [Figure 6], είτε μη-κατευθυνόμενος όταν οι ακμές δεν έχουν κατεύθυνση και μπορούν να πηγαίνουν προς οπουδήποτε μεταξύ των κόμβων [Figure 5].

Στις ακμές ενός γράφου μπορούν να επισυναπτούν βάρη, τα οποία αντιπροσωπεύουν το κόστος, την απόσταση ή άλλες χρήσιμες πληροφορίες που συνδέονται με τις σχέσεις μεταξύ των κόμβων. Τα βάρη μπορούν να χρησιμοποιηθούν για την εκτέλεση αλγορίθμων βελτιστοποίησης και την αναζήτηση των βέλτιστων μονοπατιών στον γράφο [16], [14]. Στον αλγόριθμο που θα αναλυθεί σε αυτήν την πτυχιακή εργασία τα βάρη αντιπροσοπεύουν την απόσταση της διαδρομής ή το επίπεδο της φερομόνης (θα αναλυθεί σε επόμενο κεφάλαιο).

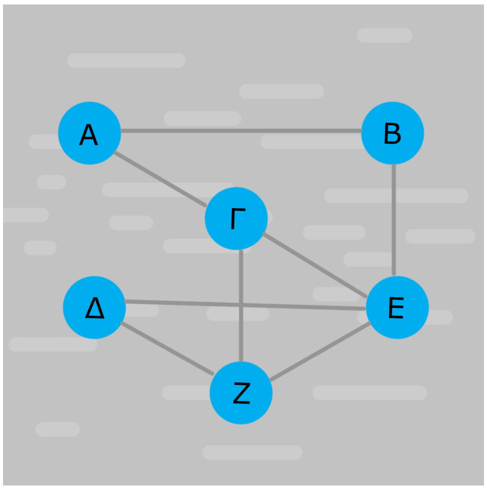

Figure 5: Μη κατευθυνόμενος γράφος.

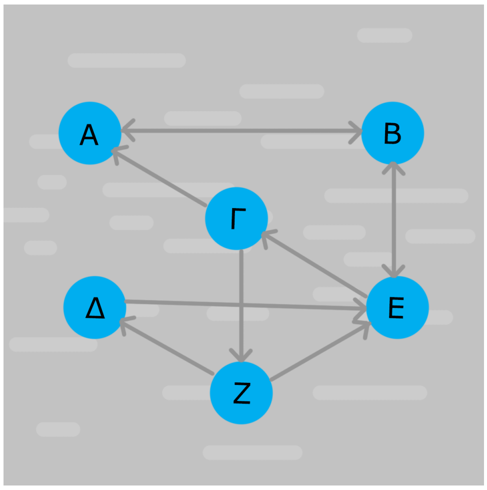

Figure 6: Κατευθυνόμενος γράφος.

2.4 Μαθηματικό υπόβαθρο

Ένας γράφος G ορίζεται από δύο σύνολα $V(G)$ και $E(G)$. Το σύνολο $V(G)$ είναι ένα πεπερασμένο σύνολο (μη άπειρο), που περιέχει ως στοιχεία τις κορυφές του γράφου. Το σύνολο $E(G)$ περιέχει τις ακμές ενός γράφου εκφρασμένες με δισύνολα δύο γειτονικών κορυφών. Έτσι, πεπερασμένος (μη - κατευθυνόμενος) γράφος, λέγεται το διατεταγμένος ζεύγος $G = (V(G), E(G))$ των πεπερασμένων συνόλων $V(G)$, $E(G)$ [16]. Αν πάρουμε ως παράδειγμα το γράφο G στο [Figure 7] παρατήρουμε ότι τα σύνολα $V(G)$ και $E(G)$ έχουν ως εξής:

- $V(G)$ =[$v1, v2, v3, v4$]
- $E(G)$ =[$e1(v1, v2), e2(v1, v3), e3(v2, v3), e4(v3, v4)$]

Επομένως ένας γράφος είναι ένα μαθηματικό αντικείμενο που ορίζεται με αυστηρό τρόπο μέσω δύο συνόλων: το σύνολο κόμβων και το σύνολο ακμών. Το σύνολο των ακμών ενος γράφου μπορεί να είναι κενό, αυτό δεν ισχύει όμως για το σύνολο των κορυφών.

2.5 Αναπαράσταση γράφων

Την κλασσική μορφή αναπαράστασης ενός γράφου την είδαμε ήδη παραπάνω [Figure 3], όμως μια τέτοια αναπαράσταση δεν είναι καθόλου πρακτική σε προγραμματιστικό επίπεδο. Για αυτό αν θέλουμε να αναπαραστίστουμε γράφους σε έναν υπολογιστή χρησιμοποιούμε δομές δεδομένων. Οι δύο πιο βασικοί μέθοδοι αναπαράστασης γράφων σε υπολογιστές είναι οι πίνακες γειτνίασης και οι λίστες γειτνίασης.

Πίνακα γειτνίασης ονομάζουμε ένα πίνακα μεγέθους nxn, όπου n ο αριθμός των κορυφών του γράφου. Κάθε κελί του πίνακα δείχνει την σχέση των αναγραφώμενων κορυφών. Σε ένα μη-κατευθυνόμενος γράφο το κελί (i, j) παίρνει την τιμή 1 αν υπάρχει η ακμή $i \longleftrightarrow j$ και 0 αν δεν υπάρχει. Είναι εύκολα αντιληπτό ότι ο πίνακας αυτός θα είναι συμμετρικός αφού η ακμή $i \longleftrightarrow j$ είναι ίδια με την ακμή $j \longleftrightarrow i$. Αντίστοιχα, σε έναν κατευθυνόμενο γράφο, το κελί (i, j) παίρνει ομοίως τιμές 0 και 1 με την διαφορά ότι ο πίνακας δεν είναι συμμετρικός, οπότε η ακμή $i \rightarrow j \neq j \rightarrow i$. Οι πίνακες γειτνίασης μπορεί να έχουν και βάρη που είναι μια επέκταση του απλού πίνακα γειτνίασης, σε αυτήν την περίπτωση, το έκαστο κελί ενός πίνακα, αντί για 1, περιέχει έναν αριθμό που ονομάζετε βάρος και υποδηλώνει κάτι ανάλογα με την χρήση του, σε έναν αλγόριθμο βελτιστοποίησης το βάρος μπορεί να υποδηλώνει την απόσταση της μιας κορυφής από την άλλη, την πιθανότητα επιλογής αυτής της διαδρομής, την επιρροή που δέχεται κάποια οντότητα σε επόμενο πιθανό πείραμα ή οποιοδήποτε άλλο κριτήριο που ανταποκρίνεται στον σκοπό του συγκεκριμένου αλγορίθμου.

Για παράδειγμα στο [Figure 7], πρόκειται για έναν μη-κατευθυνόμενο γράφο χώρις βάρη με 4 κορυφές, δηλαδή $n = 4$ και με ακμές που φαίνενται στο σύνολο $E(G)$. Επομένως ο

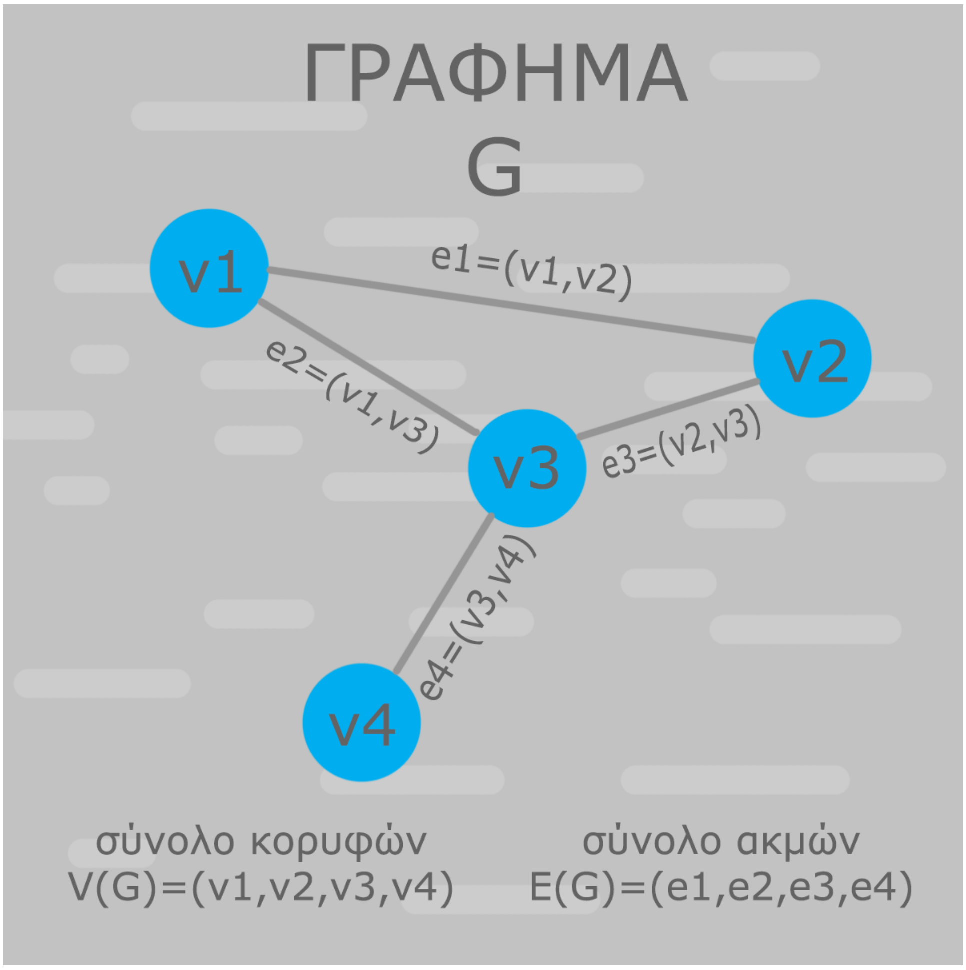

Figure 7: Σύνολα γραφήματος.

πίνακας γειτνίασης του διαμορφώνεται έτσι:

$$G_{n,n} = \begin{array}{c|cccc} & v_1 & v_2 & v_3 & v_4 \\ \hline v_1 & v_{1,1} & v_{1,2} & v_{1,3} & v_{1,4} \\ v_2 & v_{2,1} & v_{2,2} & v_{2,3} & v_{2,4} \\ v_3 & v_{3,1} & v_{3,2} & v_{3,3} & v_{3,4} \\ v_4 & v_{4,1} & v_{4,2} & v_{4,3} & v_{4,4} \end{array}$$

που αντικαθηστόντας 0-1 προκύπτει:

$$G_{4,4} = \begin{array}{c|cccc} & v_1 & v_2 & v_3 & v_4 \\ \hline v_1 & 0 & 1 & 1 & 0 \\ v_2 & 1 & 0 & 1 & 0 \\ v_3 & 1 & 1 & 0 & 1 \\ v_4 & 0 & 0 & 1 & 0 \end{array}$$

όπου 1 συμβολίζει ότι αυτές οι δύο κορυφές είναι γειτονικές έχοντας ακμή να τις ενώνει, ενώ 0 ότι δεν είναι. Με λίγα λόγια, αν το ζεύγος δύο κορυφών υπάρχει στο σύνολο $E(G)$ τότε το συγκεκριμένο κελί στον πίνακα γειτνίασης θα πάρει την τιμή 1, αλλιώς 0. Σε έναν τέτοιο πίνακα υπάρχει επανάληψη πληροφορίας καθώς το $v(i, j)$ είναι ίδιο με το $v(j, i)$. Σε περίπτωση κατευθυνόμενου γράφου όμως αυτό δε θα συνέβαινε αφού το κελί $v(i, j)$ θα συμβόλιζε αν υπάρχει ακμή από την κορυφή i προς την κορυφή j, ενώ το κελί $v(j, i)$ θα συμβόλιζε αν υπάρχει ακμή από την κορυφή j προς την κορυφή i. Σε περίπτωση ύπαρξης βαρών, τα κελιά με την τιμή 1 στον πίνακα, αντί για 1, θα είχαν την τιμή του αντίστοιχου βαρους.

Λίστα γειτνίασης ονομάζουμε μια αναπαράσταση γράφων όπου για κάθε κορυφή διατηρείται μια λίστα των γειτόνων της. Σε περίπτωση κατευθυνόμενου γράφου μπορεί να υπάρχει ξεχωριστή λίστα για τους εξερχόμενους και τους εισερχόμενους γείτονες. Αυτή η αναπαράσταση αποκτά αξία σε γράφους με αραιή συνδεσιμότητα αφού γίνεται εξοικονόμηση μνήμης.

Το πόσο μεγάλη θα είναι η λίστα εξαρτάται από τον αριθμό των κορυφών του γράφου. Κάθε στοιχείο στην λίστα συβολίζει μία ακμή του γράφου.

Στην περίπτωση έναν μη-κατευθυνόμενο γράφο, η λίστα γειτνίασης για κάθε κορυφή περιλαμβάνει τους γείτονές της, δηλαδή τις άλλες κορυφές με τις οποίες συνδέεται με μια ακμή. Αν υπάρχουν n κορυφές στον γράφο, η λίστα γειτνίασης για κάθε κορυφή περιλαμβάνει μια λίστα με το πλήθος των γειτόνων της. Για παράδειγμα στο [Figure 7] που απεικονίζεται ένας μη-κατευθυνόμενος γράφος με 4 κορυφές, άρα $n = 4$ και ακμές που φαίνενται στο σύνολο $E(G)$, αν η αναπαράσταση γινόταν με λίστα γειτνίασης θα ήταν ως εξής:

- $v1 : \{v2, v3\}$
- $v2 : \{v1, v3\}$
- $v3 : \{v1, v2, v4\}$

- $v4 : \{v3\}$

Στην περίπτωση ενός κατευθυνόμενου γράφου, σε κάθε κορυφή θα υπάρχαν 2 λίστες, μία που εκφράζει τις προηγούμενες κορυφές και μία που εκφράζει τις ακόλουθες. Για παράδειγμα στο [Figure 6] η κορυφή "Γ" σε λίστα γειτνίασης θα αναπαριστόταν ως εξής:

Γ: προηγούμενοι:$\{E\}$, ακόλουθοι:$\{A, Z\}$

Σε σχέση με τους πίνακες γειτνίασης οι λίστες γειτνίασης επιτρέπουν την εύκολη πρόσβαση στα δεδομένα του γράφου καθώς και τροποποίηση αυτών, όμως η εισαγωγή και η διαγραφή μιας ακμής είναι πιο χρονοβόρα. Επίσης σε γράφους με λίγες ακμές απαιτούν λίγοτερη μνήμη, ενώ σε πλήρη συνδεδεμένους γράφους περισσότερη.

ΚΕΦΑΛΑΙΟ 3

Α.Α.Μ. σε βάθος

Ο Αλγόριθμος Αποικίας Μυρμηγκιών (Α.Α.Μ.) είναι ένας μετευρετικός αλγόριθμος βελτιστοποίησης που έχει ως στόχο την εύρεσης βέλτιστης διαδρομής σε κάποιο πρόβλημα. Μετευρετικούς ονομάζουμε τους αλγόριθμους που είναι βασισμένοι σε ευφυείς επαναληπτικές τεχνικές και αντί να ακουλουθούν έναν αυστηρό κανόνα εξάγουν γνώση με τρόπους που είναι εμπνευσμένοι από συμπεριφορές στην φύση. [11] Συγκεκριμένα, ο Α.Α.Μ. είναι βασισμένος στην συμπεριφορά τον μυρμηγκιών για την αναζήτηση τροφής. Τα μυρμήγκια έχουν την ικανότητα να βρίσκουν πάντα την βέλτιστη διαδρομή προς την πηγή τροφής όσο δύσκολο κι αν είναι αυτό. Κατά συνέπεια κι ο Α.Α.Μ. είναι ικανός να βρει ικανοποιητικές λύσεις σε περίπλοκα προβλήματα και να εξερευνήσει μεγάλους χώρους αναζήτησης σε μικρό χρονικό διάστημα. Όπως αναφέρθηκε και στο προηγούμενο κεφάλαιο, όταν τα μυρμήγκια κινούνται στο χώρο αφήνουν φερομόνη. Αυτή η ουσία αποτελεί και τρόπο επικοινωνίας μεταξύ των μυρμηγκιών, για εύρεση βέλτιστης διαδρομής προς την τροφή, αφού όσο περισσότερη φερομόνη υπάρχει σε μία διαδρομή, τόσο αυξάναται κι η πιθανότητα ένα επόμενο μυρμήγκι να ακολουθήσει αυτή τη διαδρομή.

3.1 Βασική Θεωρία

Τα τεχνητά μυρμήγκια που χρησιμοποιούνται στον Α.Α.Μ. είναι εμπνευσμένα από την συμπεριφορά των πραγματικών μυρμηγκιών και αποτελούν διαδικασίες κατασκευής στοχαστικών λύσεων που με χρήση πιθανολογικών τεχνικών και χρησιμοποιόντας μία δυναμική δομή μνήμης που συσχετίζεται με την ποιότητα της λύσης του προηγούμενου ληφθέντος αποτελέσματος επιλύουν υπολογιστικά προβλήματα, όπως αυτό της εύρεσης βέλτιστου μονοπατιού μέσω γράφων, προσθέτοντας επαναληπτικά στοιχεία σε επιμέρους λύσεις λαμβάνοντας υπόψη ευρετικές πληροφορίες σχετικά με την επίλυση του προβλήματος και (τεχνητές) διαδρομές φερομόνης που αλλάζουν δυναμικά στο χρόνο εκτέλεσης. [5] [8]

3.2 Μαθηματικό υπόβαθρο

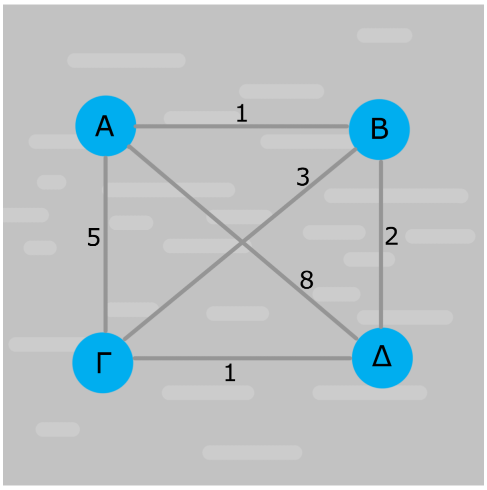

Figure 8: Απόσταση

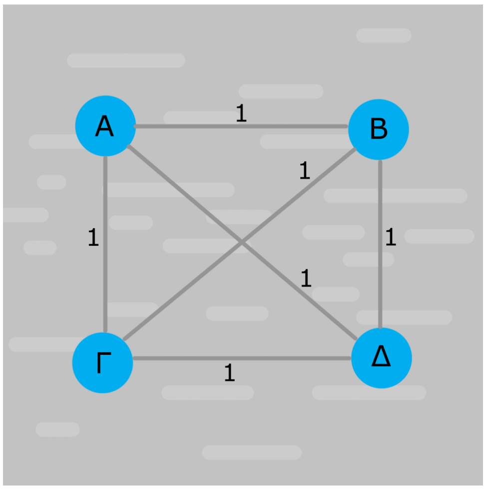

Figure 9: Φερομόνη

3.2.1 Απόσταση

Όπως αναφέρθηκε και στην δεύτερη ενότητα, ο γράφος είναι ένα απαραίτητο κομμάτι για τη μοντελοποίηση προβλημάτων με χρήση του αλγόριθμου αποικίας μυρμηγκιών. Η απόσταση, όπως και η φερομόνη, θα μοντελοποιηθούν με χρήση αναπαράστασης πίνακα. Κάθε ακμή του γράφο έχει ένα κόστος που συμβολίζει την απόσταση ή την γενικότερη ποιότητα της διαδρομής μεταξύ δύο κορυφών. Έστω οι κορυφές A, B, Γ, Δ που ενώνονται μεταξύ τους όπως φαίνεται στο [Figure 8] το οποίο αποτυπώνεται σε πίνακα γειτνίασης ως εξής:

$$Distance = \begin{array}{c|cccc} & A & B & \Gamma & \Delta \\ \hline A & 0 & 1 & 8 & 5 \\ B & 1 & 0 & 2 & 3 \\ \Gamma & 8 & 2 & 0 & 1 \\ \Delta & 5 & 3 & 1 & 0 \end{array}$$

Πρόκειται για έναν μη κατευθυνόμενο γράφο, δηλαδή συμμετρικό πίνακα και έχει βάρη σε κάθε ακμή που υποδηλώνουν το κόστος επιλογής της κάθε διαδρομής, είτε αυτό εκφράζεται ως απόσταση, είτε με κάποιον άλλο τρόπο.

3.2.2 Φερομόνη

Τα μυρμήγκια αφήνουν φερομόνη από όπου περνούν ανάλογα με την ποιότητα της λύσης που βρήκαν. Σε καλύτερες λύσεις συσσωρεύεται περισσότερη φερομόνη από τις υπόλοιπες.

Τα μονοπάτια φερομόνης εκφράζονται ως αριθμητικές πληροφορίες μέσω ενός πίνακα, που χρησιμοποιούν τα επόμενα μυρμήγκια για την κατασκευή πιθανών λύσεων στο εκάστοτε πρόβλημα και οι οποίες πληροφορίες μεταβάλονται κατά την εκτέλεση του αλγορίθμου με στόχο της εύρεση του βέλτιστου μονοπατιού. [5] Σε μονοπάτια που το μυρμήγκι επιστρέφει σε σύντομο χρονικό διάστημα συσσωρεύεται περισσότερη φερομόνη από άλλα μονοπάτια. Υπάρχουν μοντέλα που παράγωγουν περισσότερη φερομόνη ανάλογα με την ποιότητα αυτής της διαδρομής (για παράδειγμα την απόσταση, την ποσότητα της τροφής, και άλλα)

Το μαθηματικό μοντέλο αναπαράστασης της φερομόνης που εξάγει το κ-οστό μυρμήγκι στην ακμή που ενώνει τις κορυφές i και j (δηλαδή η ποσότητα της φερομόνης που παράγει) είναι αντιστρόφος ανάλογη με την απόσταση και προκύπτει από τον τύπο:

$$\Delta\tau_{i,j}^{k} = \frac{1}{L_k} \tag{1}$$

Όπου:

- L_k: Η ποιότητα της διαδρομής που βρήκε το μυρμήγκι

Έστω ότι για το πρώτο μυρμήγκι είναι παντού 1 [Figure 9]. Με αποτέλεσμα να είναι τυχαία η επιλογή διαδρομής. Στο παράδειγμα μας προκύπτει ο πίνακας:

$$Pheromone = \begin{array}{c|cccc} & A & B & \Gamma & \Delta \\ \hline A & 0 & 1 & 1 & 1 \\ B & 1 & 0 & 1 & 1 \\ \Gamma & 1 & 1 & 0 & 1 \\ \Delta & 1 & 1 & 1 & 0 \end{array}$$

Οπότε ως έδώ έχουμε ένα χάρτη με τις πιθανά μονοπάτια στον πίνακα distance και το αντίστοιχο κόστος της κάθε διαδρομής και έναν ακόμα με την "επιθυμία" του μυρμηγκιού να επιλέγει το κάθε μονοπάτι στον πίνακα pheromone.

Για να υπολογίσουμε την ποσότητα φερομόνης από μια κορυφή σε μία άλλη (χωρίς εξασθένση- θα αναλυθεί παρακάτω) υπολογίζουμε το άθροισμα της φερομόνης που εξήγαγαν τα μυρμήγκια m που πέρασαν από την κορυφή i στην j. Δηλαδή:

$$\tau_{i,j}^{k} = \sum_{k=1}^{m} \Delta\tau_{i,j}^{k} \tag{2}$$

3.2.3 Επιλογή Διαδρομής

Στο παράδειγμα μας ας υποθέσουμε ότι ένα μυρμήγκι ξεκινάει στην κορυφή A. Οι πιθανές του επιλογές όπως φαίνεται και στο [Figure 10] είναι οι B, Γ, Δ. Ποιά είναι όμως η βέλτιστη;

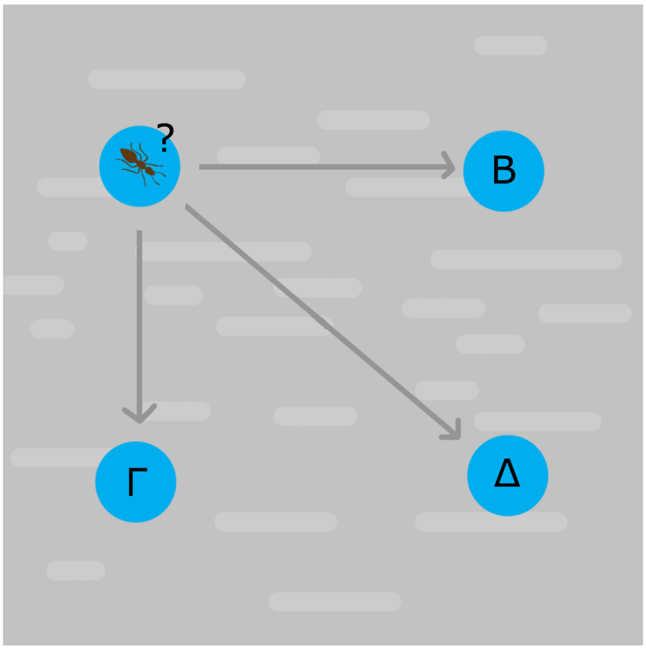

Figure 10: Πιθανές επιλογές

Με το μάτι σε ένα τόσο απλό πρόβλημα είναι εύκολο να αντιληφθούμε ποιά διαδρομή πρέπει να ακολουθήσει το μυρμήγκι, όμως αυτό δεν είναι εφικτό σε πιο περίπλοκα προβλήματα με μεγάλο αριθμό κορυφών. Έστω ότι ο αριθμός των κορυφών είναι n τότε επιλέγοντας μία κορυφη ως αρχική ο αριθμός των πιθανών κορυφών γίνεται n-1. Αφού το μυρμήγκι επιλέξει μία από αυτές μετά θα αφαιρεθεί από τις πιθανές αφού την επισκέφθηκε και θα γίνουν n-2. Έτσι θα επιλέγει μονοπάτια μεχρι να μην μείνει κανένα διαθέσιμο και να επιστρέψει στο αρχικό. Άρα ο αριθμός των πιθανών επιλογών είναι (n-1)(n-2)(n-3)...3*2*1 = (n-1)!. Όμως δεδομένου ότι διαδρομές όπως Α->Β->Γ->Δ->Α είναι ίδια με την Α->Δ->Γ->Β->Α πρέπει να μην επαναλαμβάνονται. Οπότε αφού πρόκειται για συμμετρικό πρόβλημα ο τύπος των πιθανών μονοπατιών γίνεται είναι $(n-1)!/2$ αφαιρώντας τις επαναλαμβανόμενες λύσεις όπως και στο πρόβλημα του πλανόδιου πωλητή που θα δούμε παρακάτω. Η πιθανότητα το κ-οστό μυρμήγκι να επιλέξει μια διαδρομή συμβολίζεται ως: $P^k_{i,j}$ και δίνεται από τον τύπο: [3]

$$P^k_{i,j} = \frac{(\tau_{i,j})^\alpha (\eta_{i,j})^\beta}{\sum_m (\tau_{i,m})^\alpha (\eta_{i,m})^\beta} \tag{3}$$

Όπου:

- $\tau_{i,j}$: το επίπεδο φερομόνης μεταξύ των κορυφών i και j
- $\eta_{i,j}$: η ποιότητα της διαδρομής

- m: οι υπόλοιπες διαδρομές που μπορούσε να επιλέξει το μυρμήγκι
- α, β: σταθερές που επιλέγουμε ανάλογα από την επιρροή που θέλουμε να έχει το τ και το η στην διαδικασία επιλογής. (Για παράδειγμα αν θέλουμε να επιλέξουμε μια διαδρομή βασισμένη αποκλιστικά και μόνο στο επίπεδο της φερομόνης τότε αφαιρούμε από την εξίσοση το $\eta_{i,j}$ θέτοντας το β=0).

Το γινόμενο των $\tau_{i,j} * \eta_{i,j}$ μας δίνει την "επιθυμία" του μυρμηγκιού να επιλέξει το μονοπάτι i,j. Στις περισσότερες βιβλιογραφίες αυτή η πιθανότητα αναφέρετε έτσι με διαφορετικούς συμβολισμούς στην καθεμία.

Στο παράδειγμα μας, αφού υπολογίσουμε την επιθυμία του μυρμηγκιού να επιλέξει την κάθε διαδρομή έχουμε:

- $\mathrm{AB} = \tau_{A,B}\eta_{A,B} = 1 * \frac{1}{1} = 1$
- $\mathrm{A\Gamma} = \tau_{A,\Gamma}\eta_{A,\Gamma} = 1 * \frac{1}{5} = 0.2$
- $\mathrm{A\Delta} = \tau_{A,\Delta}\eta_{A,\Delta} = 1 * \frac{1}{8} = 0.125$

Όπου οι αντίστοιχες πιθανότητες γίνονται:

- $P_{A,B} = \frac{1}{1+0.125+0.2} = \frac{1}{1.325} = 0.76$
- $P_{A,\Gamma} = \frac{0.2}{1.325} = 0.15$
- $P_{A,\Delta} = \frac{0.125}{1.325} = 0.09$

Βλέπουμε ότι πιο πιθανό είναι το μυρμήγκι να επιλέξει την διαδρομή AB όμως υπάρχει πιθανότητα να επιλέξει και τις υπόλοιπες, για την επιλογή της περιοχής χρησιμοποιώντας την πληροφορία που αντλούμε από αυτές τις πιθανότητες, αντί να χρησιμοποιήσουμε μόνο την εντολή random που μας παρέχει η python για τυχαία επιλογή θα χρησιμοποιήσουμε την τεχνική roulette wheel [7]. Υπολογίζουμε το άθροισμα συσσωρευμένο, αφού πρόκειτε για πιθανότητα το άθροισμα τους είναι 1, γίνεται η τυχαία επιλογή ενός αριθμού από το 0 έως το 1 και βρίσκουμε το σύνολο στο οποίο αντιστοιχεί αυτός ο αριθμός. Στην υλοποίησή μου η εντολή: `roulette_wheel_select()` θα επιλέγει μία πόλη τυχαία με βάση τις πιθανότητες. Ο κώδικας που υλοποιεί αυτήν την διαδικασία είναι ο παρακάτω:

```python
#roulette wheel algorithm
import random
def roulette_wheel_select(prob_array):
    total_prob = sum(prob_array)
    random_prob =random.uniform(0,total_prob)
    current=0
    for i, prob in enumerate(prob_array):
        current+=prob
        if current>random_prob:
            return i
```

```
random_prob= 0.4106305694544329
Περιοχή που επιλέχθηκε: 0 με πιθανότητα επιλογής: 0.76
```

Figure 11: Παράδειγμα roulette wheel

Όπου:

- `random_prob`: ένας τυχαίος αριθμός με τον οποίο θα επιλέξουμε περιοχή που θα πάει το μυρμήγκι.
- `current`: ο δείκτης τον περιοχών.

Στο παράδειγμά μας από 0 έως 0,76 αντιστοιχεί στην διαδρομή ΑΒ, από το 0,77 εως το 0,91 αντιστοιχεί στην διαδρομή ΑΓ και απο το 0,92 έως το 1 στην διαδρομή ΑΔ. Οπότε αν για παράδειγμα επιλεγόταν ο αριθμός 0,41 θα πηγαίναμε στην διαδρομή ΑΒ. [Figure 11]

3.2.4 Εξασθένιση Φερομόνης

Η διαδικασία της εξασθένισης φερομόνης είναι ένα σημαντικό κομμάτι για την απόδοση του αλγόριθμου και την εύρεση της βέλτιστης λύσης. Όταν όλα τα μυρμήγκια έχουν ολοκλρώσει την διαδρομή τους, ο πίνακας με τις φερομόνες πρέπει να ανανεωθεί. Αυτό γίνεται μέσω εξασθένισης της φερομόνης με ένα σταθερό ρυθμό ρ που καθορίζουμε εμείς. Ένα υψηλο ρ υποδηλώνει γρήγορη εξασθένιση ενώ ένα χαμηλό ρ αργή. Το πόσο γρήγορα εξασθενεί η φερομόνη επηρεάζει τη συμπεριφορά των μυρμηγκιών σε μεγάλο βαθμό. [4] Εμείς θα έχουμε ρ=0.5. (Όπως πρότεινεται από τους Dorigo and Stutzle) [5] Ο τύπος της εξασθένισης είναι:

$$\tau_{i,j} \leftarrow (1-\rho)\tau_{i,j}. \tag{4}$$

Αυτό επιτρέπει σε λύσεις που δεν είναι ιδανικές να μην λαμβάνονται υπόψην. Όπως αναφέρθηκε και στο κεφάλαιο 3.2.2 ο τύπος για τον υπολογισμό της φερομόνης από μία κορυφή σε μία άλλη είναι:

$$\tau_{i,j}^{k} = \sum_{k=1}^{m} \Delta\tau_{i,j}^{k} \tag{5}$$

Με εξασθένιση αυτός γίνεται:

$$\tau_{i,j}^{k} \leftarrow (1-\rho)\tau_{i,j} + \sum_{k=1}^{m} \Delta\tau_{i,j}^{k} \tag{6}$$

Όπου:

- $\tau_{i,j}$ η ποσότητα της φερομόνης στην προηγούμενη επανάληψη
- 1-ρ ο ρυθμός εξασθένισης
- $\sum_{k=1}^{m} \Delta\tau_{i,j}^{k}$ το άθροισμα φερμόνης που άφησαν όλα τα μυρμήγκια m που πέρασαν από αυτή την περιοχή

3.2.5 Διάγραμμα Αλγορίθμου

3.3 Υλοποίησή μου Αλγορίθμου σε python

Αφού κάνουμε import τις απαραίτητες βιβλιοθήκες, αρχικοποιούμε τις μεταβλητές που θα χρειαστούμε. Αυτές είναι ο αριθμός των περιοχών που μπορούν να επισκεφτούν τα μυρμήγκια (areas), ο αριθμός των μυρμηγκιών (ants), ο αριθμός των επαναλήψεων που θα εκτελεστεί ο αλγόριθμος (iterations) και οι σταθερές α και β (alpha, beta).

```python
import numpy as np
import math
#initialization of variables
areas= 5
ants= 5
iterations= 10
alpha= 1
beta= 1
```

Έπειτα δημιουργούμε τον πίνακα με τον οποίο θα μοντελοποιούμε την ποιότητα της διαδρομής (distance). Αυτό το κάνουμε τυχαία επιλέγοντας έναν αριθμό από το 0 εως το 50 για κάθε ακμή του γράφου.

```python
distance= np.random.randint(1,50,size=(areas,areas))
```

Πρόκειται για έναν μη κατευθυνόμενο γράφο οπότε πρέπει να τον μετατρέψουμε σε συμμετρτικό, αυτό το κάνουμε προσθέτοντας τον πίνακα distance με τον ανάστροφο πίνακα $distance^{Transposed}$ και διερούμε με το 2, επίσης αφαιρερούμε τους βρόχους (ακμές που ξεκινάν και τελειώνουν στην ίδια κορυφή) θέτοντας 0 στα κελιά (i,i), $i \in [0, areas)$. [10]

```python
distance= (distance + distance.T)/2
distance= distance.astype(int)
for i in range(0,areas):
    distance[i,i]=0
```

Αρχικοποιούμε τον πίνακα με τις φερομόνες (pheromone) να έχει παντού 1. Στην πρώτη επανάληψη του αλγορίθμου η επιλογή διαδρομής από ένα μυρμήγκι θα εξαρτάτε μόνο από τον πίνακα ποιότητας της διαδρομής (distance).

```python
pheromone= np.ones((areas,areas))
```

Έπειτα για όσες φορές θέλουμε να εκτελέσουμε τον αλγόριθμο (iterations): τοποθετούμε το κάθε μυρμήγκι σε μια τυχαία περιοχή ως αρχική του, αυτό το κάνουμε δημιουργόντας μια λίστα (ant_areas) μεγέθους όσα και τα μυρμήγκια και επιλέγοντας μία τυχαία περιοχή για το καθένα με χρήση της randint. [12]

```
ant_areas=[]
for ant in range(ants):
    ant_areas.append([random.randint(0,areas-1)])
print(ant_areas)
```

Η πάνω διαδικασία μπορεί επίσης να γραφεί ως εξής:

```
ant_areas = [[random.randint(0, areas-1)] for ant in range(ants)]
```

Στην συνέχεια κάθε μυρμήγκι επιλέγει την επόμενη περιοχή που θα επισκεφτεί για όσες περιοχές υπάρχουν (areas) με χρήση της φερομόνης (pheromone) και του κόστους επιλογής κάθε διαδρομής (distance). Αυτό θα το πετύχουμε μοντελοποιόντας την συνάρτηση (3). Η τυχαία επιλογή αριθμού όπως αναφέρθηκε και στο κεφάλαιο 3.2.3 θα γίνεται με την συνάρτηση roulette_wheel_select() που δημιουργήσαμε. Αρχικά υπολογίζουμε τις διαθέσιμες περιοχές που μπορεί να επισκεφθεί το κάθε μυρμήγκι (available_areas). Έπειτα υπολογίζουμε την επιθυμία του κάθε μυρμηγκιού να ακολουθήσει κάθε μονοπάτι από τα διαθέσιμα και στη συνέχεια την πίθανότητα επιλογής του κάθε μονοπατιου. Αυτή την πιθανότητα την εισάγουμε ως μεταβλητή στην συνάρτηση roulette_wheel_select() που δημιουργήσαμε και έχουμε ως αποτέλεσμα τον δείκτη που βρίσκεται η περιοχή που επιλέχθηκε να ακολουθήσει το μυρμήγκι στον πίνακα με τις διαθέσιμες περιοχές. Αυτό το ονομάζουμε next_area. Τέλος προσθέτουμε τον πίνακα με τη διαδρομή κάθε μυρμηγκιού στο ant και υπολογίζουμε το μήκος αυτών στο ant_distances.

```
for ant in ant_areas:
    while len(ant) < areas:
        area=ant[-1]
        available_areas=[a for a in range(areas) if a not in ant]
        probabilities=[(pheromone[area][a] ** alpha)*(distance[area][a] ** beta) for a
     in available_areas]
        total_pheromone=sum(probabilities)
        probabilities=[prob/total_pheromone for prob in probabilities]
        next_area=available_areas[roulette_wheel_select(probabilities)]
        ant.append(next_area)
ant_distances=[]
for ant in ant_areas:
    this_distance=0
    for i in range(len(ant)-1):
        this_distance += distance[ant[i]][ant[i+1]]
    ant_distances.append(this_distance)
print(ant_distances)
```

Έπειτα, αφού έχουμε βέλτιστη λύση από την εκτέλεση του αλγορίθμου, ανανεώνουμε τον πίνακα με τις φερομόνες έτσι ώστε καλύτερες λύσεις να είναι πιο πιθανό να επιλεγούν.

```python
for i in range(areas):
    for j in range(areas):
        if i!=j:
            pheromone[i][j] = 0.5*pheromone[i][j]+pheromone_sum[i][j]
```

Όπου το ρ=0.5, ο ρυθμός εξασθένισης και pheromone_sum το $\sum_{k=1}^{m} \Delta\tau_{i,j}^{k}$.

Αυτή την διαδικασία την επαναλαμβάνουμε όσες φορές θέλουμε να εκτελεστεί ο αλγόριθμος (iterations). Όσο μεγαλύτερο το iterations, τόσο καλύτερη θα είναι η λύση, αφού κάθε επανάληψη αυξάνει την πιθανότητα να βρεθεί μια ακόμα καλύτερη λύση από την προηγούμενη και έχει ως αποτέλεσμα να πλησιάζουμε όλο και περισσότερο στην βέλτιστη.

Παρακάτω δίνετε ο αλγόριθμος ολοκληρωμένος:

```python
#roulette wheel algorithm
import random
def roulette_wheel_select(prob_array):
    total_prob = sum(prob_array)
    random_prob =random.uniform(0,total_prob)
    current=0

    for i, prob in enumerate(prob_array):
        current+=prob
        if current>random_prob:
            return i
import numpy as np
import math
#initialization of variables
ants= 5
areas= 5
iterations= 100
alpha= 1
beta= 1
#areas graph
distance= np.random.randint(1,50,size=(areas,areas))
#convert to symmetric
distance= (distance + distance.T)/2
distance= distance.astype(int)
for i in range(0,areas):
    distance[i,i]=0
#pheromones graph
pheromone= np.ones((areas,areas))
best_one_route=[]
best_one_distance=[]
for iteration in range(iterations):
    pheromone_sum=np.zeros((areas,areas))
    #placing ants to random areas
    ant_areas = [[random.randint(0, areas-1)] for ant in range(ants)]
    for ant in ant_areas:
        while len(ant) < areas:
            #starting area of each ant
```

Μεταβλητή	**Ιδιότητα**
ants	αριθμός μυρμηγκιών
areas	αριθμός περιοχών
iterations	αριθμός επαναλήψεων
alpha	σταθερά "α"
beta	σταθερά "β"
distance	πίνακας μεγέθους areas*areas με κόστος επιλογής κάθε μονοπατιού
pheromone	πίνακας μεγέθους areas*areas με ποσότητα φερομόνης σε κάθε μονοπάτι
ant_areas	λίστα μεγέθους όσα και τα μυρμήγκια με αρχικές περιοχές για το καθένα
available_areas	διαθέσιμες περιοχές που μπορεί να επισκεφτεί το κάθε μυρμήγκι
probabilities	η "επιθυμία" επιλογής κάθε μονοπατιού από κάθε μυρμήγκι
total_pheromone	το άθροισμα όλων των επιθυμιών κάθε επανάληψης
next_area	επιλεχούσα περιοχή με χρήση roulette_wheel_select
pheromone_sum	άθροισμα φερομόνης σε κάθε μονοπάτι
ant_distances	άθροισμα κόστους διαδρομών που επέλεξε το κάθε μυρμήγκι
best_route	λίστα με καλύτερες διαδρομές της κάθε επανάληψης
best_one_route	μονοπάτι καλύτερης διαδρομής από όλες τις επαναλήψεις
best_one_distance	κόστος καλύτερης διαδρομής

Table 1: Πίνακας με μεταβλητές αλγόριθμου

```
            area=ant[-1]
            available_areas=[a for a in range(areas) if a not in ant]
            #calculating the desire of the ant choosing each path
            probabilities=[(pheromone[area][a] ** alpha)*(1/distance[area][a]  ** beta) for a in available_areas]
            #calculating the sum of all desires
            total_pheromone=sum(probabilities)
            probabilities=[prob/total_pheromone for prob in probabilities]
            #choosing next path using roulette wheel selection
            next_area=available_areas[roulette_wheel_select(probabilities)]
            pheromone_sum[area][next_area] = 0.5*pheromone_sum[area][next_area]+1/distance[area][next_area]
            pheromone_sum[next_area][area] = 0.5*pheromone_sum[next_area][area]+1/distance[next_area][area]
            ant.append(next_area)
    #distance sum of the route each ant have chosen
    ant_distances=[]
    for ant in ant_areas:
        this_distance=0
        for i in range(len(ant)-1):
            this_distance += distance[ant[i]][ant[i+1]]
        ant_distances.append(this_distance)
    #the best route of all the chosen ones
    best_route=ant_areas[ant_distances.index(min(ant_distances))]
    best_one_route.append(best_route)
    best_one_distance.append(min(ant_distances))
    #updating pheromones
    for i in range(areas):
        for j in range(areas):
            if i!=j:
                pheromone[i][j] = 0.5*pheromone[i][j]+pheromone_sum[i][j] #evaporation rate=0.5
print("best route after ", iteration+1, " iterations:", min(best_one_route), "with distance (cost) summation:", min(best_one_distance))
```

3.4 Εκτέλεση του Αλγορίθμου

Στο παράδειγμα που εκτελέσαμε τον αλγόριθμο έχουμε ως δεδομένα 5 μυρμήγκια που εκτελούν αναζήτηση μεταξύ 5 περιοχών 10 φορές με alpha και beta 1, η εκτέλεση του αλγορίθμου με τα παραπάνω δεδομένα δίνουν αποτέλεσμα παρόμοιο με αυτό που φαίνεται στο [Figure 12].

Ανάλογα από τις απαιτήσεις του εκάστοτε προβλήματος, οι μεταβλητές μπορούν να αλλάξουν, τί γίνεται όταν αλλάξουμε τα alpha και beta, τί συμβαίνει με μικρό αριθμό επαναλήψεων, πώς λειτουργεί το πρόγραμμα με εξασθένιση και πώς χωρίς εξασθένιση, πώς σχετίζεται η ποιότητα της λύσης με τον αριθμό των μυρμηγκιών; Ερωτήσεις σαν κι αυτές θα απαντηθούν παρακάτω με παραδείγματα εκτέλεσης του αλγορίθμου και με τα αντίστοιχα αποτελέσματα.

```
best route after  10  iterations: [0, 1, 3, 2, 4] with distance (heuristic function) summation: 69
```

Figure 12: Παράδειγμα με: ants=5, areas=5, iterations=10, alpha=beta=1.

```
[0, 10, 20, 30, 40]
[10, 0, 10, 20, 30]
[20, 10, 0, 10, 20]
[30, 20, 10, 0, 10]
[40, 30, 20, 10, 0]
```

Figure 13: Πίνακας distance που θα χρησιμοποιηθεί για τα παραδείγματα.

Για εύκολη κατανόηση του προβλήματος και για τον σκοπό της επίτευξης των συγκρίσεων θα γίνει χρήση ίδιου πίνακα distance σε όλα τα παρακάτω παραδείγματα, ο οποίος δίνεται στο [Figure 13] με βέλτιστα αποτελέσματα που φαίνονται στο [Figure 14]

3.4.1 Τροποποίηση των alpha, beta

Σε περίπτωση που το alpha πάρει την τιμή 0, η επιρροή που έχει ο πίνακας με τις φερομόνες (pheromone) μηδενίζεται, αφαιρόντας έτσι το δυνατότητα του αλγόριθμου να "μάθει" από προηγούμενες εκτελέσεις, με αποτέλεσμα ο αλγόριθμος να βασίζεται μόνο στο πίνακα με το κόστος επιλογής κάθε διαδρομής (distance) και κατά συνέπεια στο ποιό είναι το καλύτερο μονοπάτι εκείνη την χρονική στιγμή χωρίς να δίνεται βάση σε προηγούμενες λύσεις. Το πρόγραμμα μας εξάγει πάλι βέλτιστα αποτελέσματα [Figure 14], αλλά παρατηρώντας τα, είναι εύκολα αντιληπτό ότι η λύση βρέθηκε λόγο της απλότητας του προβλήματος.

Αντίστοιχα, αν το beta πάρει την τιμή 0, μηδενίζεται η επιρροή του πίνακα το κόστος επιλογής κάθε διαδρομής (distance) και το αποτέλεσμα θα βασίζεται καθαρά στον πίνακα με τις φερομόνες, πάλι δύνονται βέλτιστα αποτελέσματα, αυτό συμβαίνει γιατί έχουμε θέσει τον πίνακα με τις φερομόνες να είναι παντού 1, οπότε η πιθανότητα σε 10 επαναλήψεις να βρει την βέλτιστη διαδρομή είναι μεγάλη, αν γίνει αλλαγή του πίνακα με τις φερομόνες αυξάνοντας την

```
best route after  10  iterations: [0, 1, 2, 3, 4] with distance (cost) summation: 40
```

Figure 14: Βελτιστα αποτελέσματα για είσοδο distance του Figure 13.

```
[0, 1, 2, 3, 400]
[1, 0, 1, 2, 3]
[2, 1, 0, 1, 2]
[3, 2, 1, 0, 1]
[400, 3, 2, 1, 0]
```

Figure 15: Πίνακας pheromone με διαφορετικές επιρροές.

```
best route after  10  iterations: [0, 4, 3, 1, 2] with distance (cost) summation: 70
```

Figure 16: Αποτελέσματα για pheromone [Figure 15] και beta=0.

επιρροή μιας μη βέλτιστης διαδρομής τότε παρατηρούμε ότι ενώ με alpha=beta=1 καταφέρνει να ανακάμψει γρήγορα και δίνεται πάλι η βέλτιστη διαδρομή ως αποτέλεσμα [Figure 14], στην περίπτωση που το beta=0 αυτό δεν συμβαίνει τόσο εύκολα [Figure 16].

Παρατηρούμε ότι τα alpha και beta αποτελούν συμαντικοί παράμετροι για την εύρεση λύσης καθώς επηρεάζουν το πως ο αλγόριθμος βρίσκει την βέλτιστη διαδρομή, αν είναι ιδανικά για το εκάστοτε πρόβληματα τότε η λύση θα βρεθεί γρηγορότερα. Κατά συνέπεια αν alpha>beta τα μυρμήγκια εστειάζουν στις προηγούμενες λύσεις, δίνοντας έμφαση στην "μνήμη" του προγράμματος, ενώ αν beta>alpha έμφαση δίνετε στην αξία της λύσης την συγκεκριμένη χρονική στιγμή. Κατά συνέπεια η ισορροπία μεταξύ αυτών των δύο παραμέτρων και η σωστή ρύθμιση τους είναι κομβικής σημασίας για την επίτευξη καλών αποτελεσμάτων.

3.4.2 Εξασθένιση/Ρυθμός εξασθένισης

Άλλος ένα σημαντικός παράγοντας για την ορθή εκτέλεση του αλγόριθμου της αποικείας των μυρμηγκιών είναι ο ρυθμός εξασθένισης. Η ρυθμός αυτός βοηθάει στην εξάλειψη των ίχνων φερομόνης σε μη ιδανικά μονοπάτια, που πιθανών οδηγήσουν σε λάθος αποτελέσματα αφού θα αποπροσανατολίσουν τα μυρμήγκια, χωρίς όμως να έχουμε απώλεια πληροφορίας. [9]. Ένας υψηλός ρυθμός εξασθένισης (ρ) υποδηλώνει ότι η φερομόνη εξασθενεί ταχύτερα με αποτέλεσμα να παίζει συμαντικότερο ρόλο το κόστος επιλογής κάθε διαδρομής και οι πληροφορίες που σχετίζονται με την φερομόνη από προηγούμενες επαναλήψεις να χάνονται πιο γρήγορα. Δηλαδή, δίνεται περισσότερη βάση στην εξερεύνηση του διαθέσιμου χώρου. Αντίθετα ένας πολύ χαμηλός ρυθμός εξασθένισης (ρ) υποδηλώνει ότι η φερομόνη θα παραμένει στις ακμές περισσότερο χρόνο με αποτέλεσμα τα μυρμήγκια να ακολουθούν τα ίδια μονοπάτια που εξερευνήθηκαν νωρίτερα. Συνοψίζοντας, υψηλός ρυθμός εξασθένισης δίνει έμφαση στην εξερεύνιση του χώρου ενώ χαμηλός ρυθμός δίνει έμφαση στην "μνήμη" του προγράμματος.

```
[[ 0  1 20 30 40]
 [ 1  0  1 20 30]
 [20  1  0  1 20]
 [30 20  1  0  1]
 [40 30 20  1  0]]
```

Figure 17: Πίνακας pheromone με διαφορετικές επιρροές.

```
best route after  1  iterations: [0, 3, 1, 2, 4] with distance (cost) summation: 80
```

Figure 18: Αποτελέσματα για iterations=1.

Σε περίπτωση που μηδενίσουμε την εξασθένιση παρατηρούμε ότι η σύγκλιση σε βέλτιστη λύση συνήθως καθυστερεί, αυτό συμβαίνει γιατί τα μυρμήγκια τείνουν να επιλέγουν τα ίδια μονοπάτια ακόμα κι αν δεν είναι βέλτιστα. Στο παραδειγμα μας οδηγεί πάλι στην βέλτιστη λύση [Figure 14] αλλά χρειάζεται (συνήθως) περισσότερο αριθμό επαναλήψεων.

3.4.3 Επαναλήψεις

Ακόμα ένας σημαντικός παράγοντας για την εύρεση βέλτιστης λύσης είναι ο αριθμός επαναλήψεων (ιτερατιονς) που θα εκτελεστεί ο αλγόριθμος και τα μυρμήγκια θα ψάξουν για λύση. Θεωρόντας τον πίνακα με την φερομόνη μη ιδανικό [Figure 17], δοκιμάζουμε τα εξής σενάρια:

- iterations=1 [Figure 18]: Σε αυτή την περίπτωση, τα μυρμήγκια επηρεασμένα από την φερομόνη ακολουθούν μη βέλτιστη διαδρομή και δεν τους δίνεται χρόνος για εξασθένιση, με αποτέλεσμα να οδηγούμαστε πάντα σε λάθος λύση.

- iterations=10 [Figure 19]: Αν αυξήσουμε τον αριθμό επαναλήψεων, παρατηρούμε ότι δίνεται χρόνος στα μυρμήγκια να ανακάμψουν από την λάθος διαδρομή και να βελτιωθεί η ποιότητα της λύσης (πολλές φορές και στην βέλτιστη για το παραδειγμα μας), αυτό συμβαίνει αφού δίνεται ο απαραίτητος χρόνος για αντικατάσταση του πίνακα με τις φερομόνες και εξασθένιση της λανθασμένης διαδρομής.

- iterations=100 [Figure 20]: Αν αυξήσουμε ακόμα περισσότερο τον αριθμό επαναλήψεων, παρατηρούμε ότι τα μυρμήγκια όσες φορές και να εκτελέσουμε τον αλγόριθμο πάντα καταφέρνουν να βρουν την βέλτιστη λύση στο συγκεκριμένο παράδειγμα.

```
best route after  10  iterations: [3, 4, 2, 1, 0] with distance (cost) summation: 50
```

Figure 19: Αποτελέσματα για iterations=10.

```
best route after  100  iterations: [0, 1, 2, 3, 4] with distance (cost) summation: 40
```

Figure 20: Αποτελέσματα για iterations=100.

3.4.4 Αριθμός μυρμηγκιών

Ένας ακόμα σημαντικός παράγωντας στον αλγόριθμο μας είναι ο αριθμός των μυρμηγκιών. Αυτός ο αριθμός μπορεί να επηρεάζει σημαντικά την απόδοσή του, μεγάλος αριθμός μυρμηγκιών υποδηλώνει μεγαλύτερο εύρως εξερεύνησης σε λιγότερες επαναλήψεις καθώς και γρηγορότερη σύγκλιση σε μια αποδεκτή λύση, βέβαια αυξάνετε το κόστος σε υπολογιστική ισχύ κάνοντας τον πρόγραμμα πιο αργό στην εκτέλεση του. Αντίστοιχα, μικρός αριθμός μυρμηγκιών μπορεί να όδηγήσει σε πιο αργή σύγκλιση της λύσης στην βέλτιστη, αφού η διαδικασία εξερεύνησης θα είναι πιο χρονοβόρα. Στο παράδειγμά μας, μικρός αριθμός μυρμηγκιών αρκεί για εξερεύνηση των 5 περιοχών, όσο πιο περίπλοκο γίνεται το πρόβλημα, με περισσότερες περιοχές για εξερεύνηση τόσο περίσσοτερα θα πρέπει να είναι και τα μυρμήγκια.

3.5 Εφαρμογές Αλγορίθμου

Ο αλγόριθμος της αποικείας των μυρμηγκιών, όπως ήδη αναφέρθηκε, δίνει λύση σε πληθώρα προβλημάτων του πραγματικού μας κόσμο, όπως είναι η εύρεση βέλτιστης διαδρομή σε προβλήματα δομολόγησης, η βελτιστοποίηση των δικτύων επικοινωνίας, η δρομολόγηση δικτύου δεδομένων και πολλά άλλα.

3.5.1 Πρόβλημα του πλανόδιου πωλητή

Ένα από το πιο μελετημένα προβλήματα που εφαρμόζεται αυτός ο αλγόριθμος είναι αυτό του πλανόδιου πωλητή (Traveling Salesman Problem - TSP), όπου έχει ως στόχο την εύρεση της βέλτιστης διαδρομής που θα περιέρχει όλες τις πόλεις ενός χάρτη ακριβώς μία φορά και θα επιστρέφει στην αρχική πόλη.

Για την επίτευξη αυτού με χρήση του αλγόριθμου μας, πρέπει να γίνει προσθήκη της επιστροφής του μυρμηγκιού στην αρχική κορυφή και έπειτα εύρεση της βέλτιστης διάδρομή. Εύκολα επιτυγχάνεται αυτό με μια μικρή προσθήκη στον κώδικα. Μετά το βήμα "#distance

```
best route after  100  iterations: [0, 1, 3, 2, 4, 0] with distance (cost) summation: 79
```

Figure 21: Εκτέλεση με επιστροφή μυρμηγκιού στην αρχική του κορυφή.

sum of the route each ant have chosen" και πριν το "#the best route of all the chosen ones", γίνεται προσθήκη του "#returning each ant to its original area", όπως δίνεται παρακάτω:

```python
#returning each ant to its original area
for i in range(len(ant_areas)):
    ant_distances[i] += distance[ant_areas[i][0]][ant_areas[i][4]]
    ant_areas[i].append(ant_areas[i][0])
```

Παράδειγμα εκτέλεσης αυτού του αλγορίθμου δίνεται στο [Figure 21].

Γίνεται εύκολα κατανοητό ότι το πρόβλημα του πλανόδιου πωλήτη, που ξεκινάει από την πόλη όπου βρίσκεται με στόχο να περάσει όλες τις πόλεις με πελάτη και να γυρίσει στην αρχική το συντομότερο δυνατό επισκέπτοντας κάθε πόλη με πελάτη μόνο μία φορά, ο αλγόριθμος της αποικείας των μυρμηγκιών μπορεί να το προσομοιώσει ικανοποιητικά. Αυτό γίνεται αν υποθέσουμε ότι ο γράφος $G = (V(G), E(G))$ που είδαμε παραπάνω αντιπροσωπεύει ένα χάρτη με πόλεις ($V(G)$) και δρόμους ($E(G)$). (Σημείωση ότι εάν το γράφημα δεν είναι πλήρες, δήλαδη δεν υπάρχουν δρόμοι που να ενώνουν όλες τις πόλεις, προσθέτουμε ακμές έτσι ώστε να γίνει με βάρος αρκετά μεγάλο καθιστώντας το απίθανο να επιλεγεί ως βέλτιστη λύση, όπως προτείνεται από Dorigo και Stützle στο [6]). Σε κάθε δρόμο ($E(G)$) ανατίθεται και ένα κόστος που αντιπροσωπεύει την απόσταση μεταξύ δύο πόλεων (κόστος), αυτό μπορούμε να το προσαρόσουμε ανάλογα με την επιθυμία του πωλητή να ακολουθήσει μία διαδρομή (αν για παραδειγμα ο δρόμος είναι ανηφορικός και είναι με τα πόδια), την κίνηση σε κάθε δρόμο (αν είναι με όχημα) και από πολλούς άλλους παράγοντες. Για λόγους απλότητας θα χρησιμοποιηθεί μόνο η απόσταση. Ο στόχος στο πρόβλημα του πλανόδιου πωλητή είναι η εύρεση του ελάχιστου σε απόσταση κύκλου Hamilton στο γράφο-χάρτη, ένας κύκλος Hamilton είναι ένα μονοπάτι που που επισκέφτεται κάθε πόλη μία φορά και καταλήγει στην αρχική. [6]

Για την εκτέλεση του αλγορίθμου που υλοποιήθηκε παραπάνω με στόχο την επίλυση του προβλήματος του πλανόδιου πωλητή πρέπει ο πίνακας με το κόστος κάθε διαδρομής (distance) να αντιπροσωπεύει πόλεις και τις αντίστοιχες αποστάσεις μεταξύ τους.

Διάγραμμα ροής προβλήματος:

References

[1] Christian Blum. Ant colony optimization: Introduction and recent trends. *Physics of Life reviews*, 2(4):353–373, 2005.

[2] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm intelligence: from natural to artificial systems*. Oxford university press, 1999.

[3] Chirag Chandrashekar, Pradeep Krishnadoss, Vijayakumar Kedalu Poornachary, Balasundaram Ananthakrishnan, and Kumar Rangasamy. Hwacoa scheduler: Hybrid weighted ant colony optimization algorithm for task scheduling in cloud computing. *Applied Sciences*, 13(6):3433, 2023.

[4] Laurence Dawson and Iain Stewart. Improving ant colony optimization performance on the gpu using cuda. In *2013 IEEE Congress on Evolutionary Computation*, pages 1901–1908. IEEE, 2013.

[5] Marco Dorigo and Thomas Stützle. The ant colony optimization metaheuristic: Algorithms, applications, and advances. *Handbook of metaheuristics*, pages 250–285, 2003.

[6] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization*. The MIT Press, 2004.

[7] Adam Lipowski and Dorota Lipowska. Roulette-wheel selection via stochastic acceptance. *Physica A: Statistical Mechanics and its Applications*, 391(6):2193–2196, 2012.

[8] Michalis Mavrovouniotis, Changhe Li, and Shengxiang Yang. A survey of swarm intelligence for dynamic optimization: Algorithms and applications. *Swarm and Evolutionary Computation*, 33:1–17, 2017.

[9] Michalis Mavrovouniotis and Shengxiang Yang. Ant colony optimization with self-adaptive evaporation rate in dynamic environments. In *2014 IEEE symposium on computational intelligence in dynamic and uncertain environments (CIDUE)*, pages 47–54. IEEE, 2014.

[10] Rubén Pérez Sanz. Introduction to linear algebra. https://rubencioak.github.io/Teaching/Math_Brush_up_2020/Math%20-%20Linear%20Algebra.pdf, 2020. Accessed: 2023-08-19.

[11] Celso C Ribeiro, Pierre Hansen, Vittorio Maniezzo, and Antonella Carbonaro. Ant colony optimization: an overview. *Essays and surveys in metaheuristics*, pages 469–492, 2002.

[12] w3schools. Python tutorial. https://www.w3schools.com/python/default.asp. Accessed: 2023-08.

[13] Ιωάννης Γεωργιάδης. Θεωρία Γράφων και αλγόριθμοι εύρεσης βέλτιστης διαδρομής σε δίκτυα. Master's thesis, Δημοκρίτειο Πανεπιστήμιο Θράκης, 2017.

[14] Γεώργιος Γκέρτσης. Θεωρία Γράφων Και Εύρεση Βέλτιστης Διαδρομής. Master's thesis, Δημοκρίτειο Πανεπιστήμιο Θράκης, 2023.

[15] Ιωάννης Μανωλόπουλος. Θεωρία και Αλγόριθμοι Γράφων. http://eclass.auth.gr/courses/OCRS264/, 2014. Accessed: 2023-8-07.

[16] Ιωάννης Ντενησιώτης. Θεωρία Γράφων και Εφαρμογές της. Master's thesis, Δημοκρίτειο Πανεπιστήμιο Θράκης, 2021.

Εγκατάσταση και εκτέλεση Anaconda διανομής και jupyter notebook

Το anaconda είναι μια διανομή ανοιχτού κώδικα, στην οποία περιλαμβάνεται και το jupyter notebook για εκτέλεση κώδικα σε python, παράλληλα με άλλα χρήσιμα εργαλεία. Κατεβάζουμε την διανομή από την ιστοσελίδα τους (`https://www.anaconda.com/download`) ανάλογα με το λειτουργικό που χρησιμοποιούμε, και το τρέχουμε. Κατεβάζουμε με τα default settings. Τρέχουμε το πρόγραμμα και θα μας ανοίξει το anaconda navigator με πολλά διαθέσιμα εργαλεία, θα υπάρχει και το jupyter notebook, το κατεβάζουμε και ανοιγουμε ένα new jupyter notebook. Θα μας ανοίξει ένα καινούργιο παράθυρο στο οποίο μπορούμε να γράψουμε κώδικα σε γλώσσα python. Εκεί εκτελούμε όλους τους αλγόριθμους που παρουσιάζονται σε αυτήν την εργασία.

Ολοκληρωμένος κώδικας

Ο κώδικα που παρουσιάστηκε σε αυτήν την πτυχιακή μπορεί να βρεθεί στο παρακάτω link: https://github.com/vaggbik/ant_colony_algorithm και δίνεται ολοκληρωμένος παρακάτω:

```python
#roulette wheel algorithm
import random
def roulette_wheel_select(prob_array):
    total_prob = sum(prob_array)
    random_prob =random.uniform(0,total_prob)
    current=0

    for i, prob in enumerate(prob_array):
        current+=prob
        if current>random_prob:
            return i
import numpy as np
import math
#initialization of variables
ants= 5
areas= 5
iterations= 10
alpha= 0.5
beta= 0.5
#areas graph
distance= np.random.randint(1,50,size=(areas,areas))
#convert to symmetric
distance= (distance + distance.T)/2
distance= distance.astype(int)
for i in range(0,areas):
    distance[i,i]=0
#pheromones graph
pheromone= np.ones((areas,areas))
best_one_route=[]
best_one_distance=[]
for iteration in range(iterations):
    pheromone_sum=np.zeros((areas,areas))
    #placing ants to random areas
```

```python
    ant_areas = [[random.randint(0, areas-1)] for ant in range(ants)]
    for ant in ant_areas:
        while len(ant) < areas:
            #starting area of each ant
            area=ant[-1]
            available_areas=[a for a in range(areas) if a not in ant]
            #calculating the desire of the ant choosing each path
            probabilities=[(pheromone[area][a] ** alpha)*(1/distance[area][a]  ** beta) for a in available_areas]
            #calculating the sum of all desires
            total_pheromone=sum(probabilities)
            probabilities=[prob/total_pheromone for prob in probabilities]
            #choosing next path using roulette wheel selection
            next_area=available_areas[roulette_wheel_select(probabilities)]
            pheromone_sum[area][next_area] = 0.5*pheromone_sum[area][next_area]+1/distance[area][next_area]
            pheromone_sum[next_area][area] = 0.5*pheromone_sum[next_area][area]+1/distance[next_area][area]
            ant.append(next_area)
    #distance sum of the route each ant have chosen
    ant_distances=[]
    for ant in ant_areas:
        this_distance=0
        for i in range(len(ant)-1):
            this_distance += distance[ant[i]][ant[i+1]]
        ant_distances.append(this_distance)

    #returning each ant to its original area
    for i in range(len(ant_areas)):
        ant_distances[i] += distance[ant_areas[i][0]][ant_areas[i][4]]
        ant_areas[i].append(ant_areas[i][0])

    #the best route of all the chosen ones
    best_route=ant_areas[ant_distances.index(min(ant_distances))]
    best_one_route.append(best_route)
    best_one_distance.append(min(ant_distances))
    #updating pheromones
    for i in range(areas):
        for j in range(areas):
            if i!=j:
                pheromone[i][j] = 0.5*pheromone[i][j]+pheromone_sum[i][j] #evaporation rate=0.5
print("\nbest route after ", iteration+1, " iterations:", min(best_one_route), "with distance (cost) summation:", min(best_one_distance),"\n")
```