Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Πληροφορική»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Ανάπτυξη λογισμικού οπτικοποίησης γράφων μεγάλης κλίμακας με χρήση παραλληλότητας και αλγορίθμων διάταξης** **Large scale graph visualization software using parallelization and distribution algorithms**
Ονοματεπώνυμο Φοιτητή	**Ορέστης Μωρέσης**
Πατρώνυμο	**Χρήστος**
Αριθμός Μητρώου	**ΜΠΠΛ/ 14060**
Επιβλέπων	**Τσιχριτζής Γεώργιος, Καθηγητής**

Ημερομηνία Παράδοσης **Οκτώβριος 2018**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Τσιχριτζής Γ., Καθηγητής	Σακκόπουλος Ε., Επίκουρος Καθηγητής	Σωτηρόπουλος Δ., Διδάκτωρ

Περίληψη

Τα δεδομένα καταλαμβάνουν διαρκώς μεγαλύτερο χώρο και γίνονται όλο και πιο χαοτικά με αποτέλεσμα οι παλαιότερες, πιο παραδοσιακές μορφές αποθήκευσης και αναπαράστασης να καθιστούν δυσκολότερη την κατανόησή τους και την επεξεργασία τους. Πολλά σύγχρονα συστήματα έχουν χρησιμοποιηθεί στο πεδίο χρήσης όπως οι Βάσεις Δεδομένων NOSQL και Graph Database. Οι βάσεις δεδομένων γραφημάτων είναι η καλύτερη μορφή απεικόνισης λογικών συνδέσεων γνωστών και ως γραφήματα δικτύου. Αυτό ξεδιπλώνει ένα τεράστιο πεδίο χρήσεων και εφαρμογή νέων αλγορίθμων. Το αντικείμενο αυτού του έργου είναι να επιθεωρήσει την μεταχείριση και την αναπαράσταση γράφων από ένα χαμηλό επίπεδο προγραμματισμού και να δοκιμάσει νέες μορφές αποθήκευσης καθώς και σύγχρονες γλώσσες προγραμματισμού αλλά και τη συνεργασία μεταξύ τους αναπτύσσοντας ένα λογισμικό που εισάγει ακατέργαστα δεδομένα, με τελικό σκοπό την προεπισκόπισή τους χρησιμοποιώντας μερικούς αλγορίθμους διάταξης και των απλών αλλά γρήγορων γραφικών μεθόδων.

Abstract

Data is getting constantly larger and more chaotic in such a way that older, more traditional forms of storage and representation render it more difficult to understand and process it. A lot of modern systems have been in the scope of use such as NOSQL and Graph Databases. Graph databases are the best form of representing logical connections also known as network graphs. This unfolds a vast field of uses and new algorithms. The point of this project is to inspect graph manipulation and representation from a low level of programming and try new forms of storage as well as modern programming languages and their cooperation between each other by developing a software that imports raw data and previews them using a couple of distribution algorithms and simple but fast graphics.

Περιεχόμενα

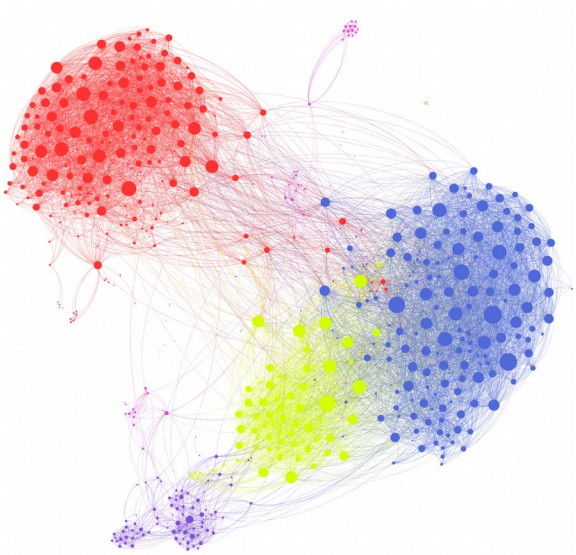
I. Εισαγωγή

1 Γράφοι ή γραφήματα

Η εξόρυξη δεδομένων έχει μακρά ιστορία, η οποία χωρίζεται σε πολλούς τομείς, όπως σχεδιασμός βασικών συστημάτων, τα στατιστικά στοιχεία, η αναγνώριση προτύπων, η εκμάθηση μηχανών και η προβολή δεδομένων. Αυτή είναι στην πραγματικότητα η διαδικασία εντοπισμού μοντέλων προεπισκόπησης από τα δεδομένα που αντλήθηκαν. Η εξόρυξη γραφημάτων είναι ένας πιο σύγχρονος κλάδος της εξόρυξης δεδομένων και είναι ουσιαστικά η διαδικασία συλλογής και ανάλυσης δεδομένων που μπορούν να αναπαρασταθούν με τη μορφή γράφων, δηλαδή γραφημάτων που δεν χρησιμοποιούν το καρτεσιανό σύστημα προβολής, αλλά μια πιο αφηρημένη αναπαράσταση στοιχείων στο μορφή κόμβων και οποιαδήποτε λογική σύνδεση μεταξύ τους υπό τη μορφή αιχμών, κυρίως σε δύο αλλά μερικές φορές σε τρεις διαστάσεις.

Αξίζει να σημειωθεί πως στην μαθηματική ορολογία υπάρχει διαχωρισμός μεταξύ της γραφικής παράστασης plot και του γραφήματος graph διότι είναι εύκολο να δημιουργηθεί σύγχυση μεταξύ των δύο. Από εδώ και στο εξής όποτε αναφέρεται ο όρος γράφημα θα αφορά σε ένα graph το οποίο φτιάχνεται συνήθως όταν πρέπει να αναπαρασταθούν οι σχέσεις μεταξύ των στοιχείων ενός πεπερασμένου συνόλου. [1]
Αυτός ο τρόπος αναπαράστασης βοήθησε στην επίλυση πολλών προβλημάτων στη βιολογία, τη χημεία, τη διαχείριση και επικοινωνία επιχειρήσεων και στα δίκτυα. Αν και η γραφική εξόρυξη και ανάλυση είναι ένα σχετικά νέο πεδίο, βασίζεται σε μια σταθερή βάση της κλασσικής θεωρίας γραφημάτων, της εκτίμησης κόστους, καθώς και κοινωνιολογικών εννοιών όπως τα πρότυπα συσχετισμού ανθρώπων ή ομαδοποιήσεων.

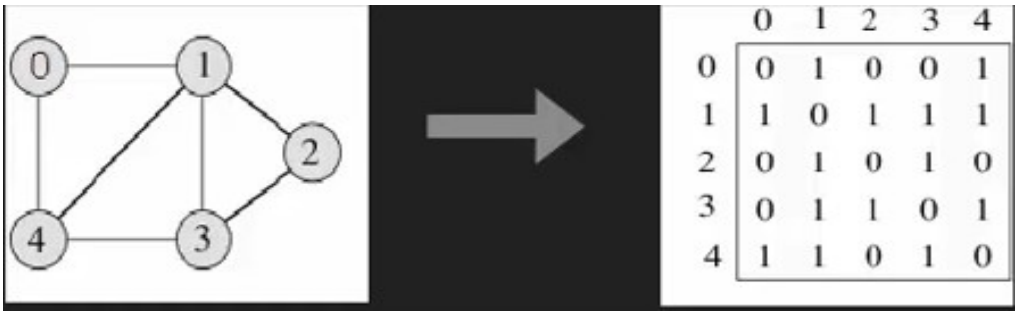
Εν κατακλείδι, τα γραφήματα είναι συχνά πολύ επιτυχημένα παραδείγματα μαθηματικής μοντελοποίησης. Παρά την απλότητα τους, βοηθούν ιδιαίτερα στην περιγραφή και τη μελέτη πολλών ενδιαφερόντων και πραγματικών προβλημάτων. Επιπλέον προσφέρουν χρήσιμες εφαρμογές και συνδέσεις των μαθηματικών με την καθημερινή ζωή, κάτι που είναι πολύ σημαντικό. Δουλεύοντας με γραφήματα υπάρχει τριβή με καθαρά μαθηματικές συλλογιστικές μορφές υψηλής εκπαιδευτικής αξίας. Ως παράδειγμα θα μπορούσαν να αναφερθούν η επαγωγική η συνδυαστική, η χωρική συλλογιστική, κτλ. [1]



Εικ. 1: Γράφημα κοινωνικού δικτύου [12]

a) Τρόποι αναπαράστασης δεσμών

Adjacency Matrix: Πίνακας δύο διαστάσεων με V κόμβους σε πίνακα διαστάσεων VxV δυαδικών τιμών 0-1. Aij = 1 σημαίνει ότι ο κόμβος I με τον κόμβο j συνδέονται. Στην περίπτωση που έχουμε undirected graph ο πίνακας αυτός είναι συμμετρικός, το αντίθετο συμβαίνει σε directed graph. [2]



Θετικά:

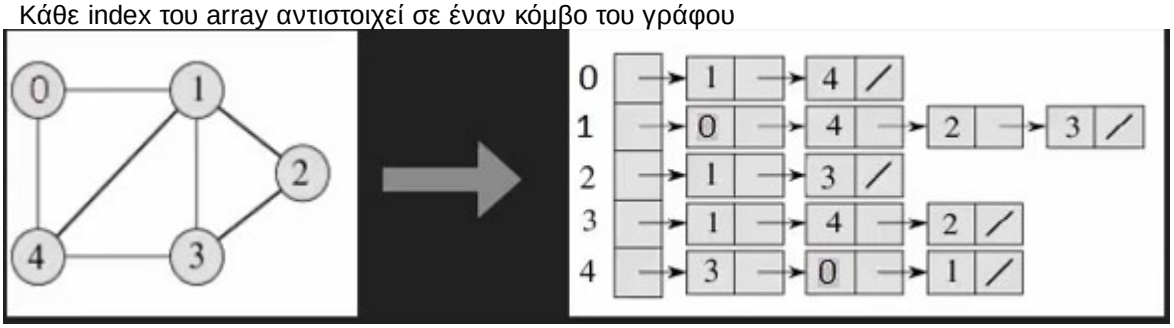
- Εύκολα υλοποιήσιμος
- Η αφαίρεση ενός κόμβου χρειάζεται O(1) χρόνο
- Queries ύπαρξης ακμών μεταξύ ορισμένων κόμβων είναι αποδοτικά και μπορούν να εκτελεστούν σε χρόνο O(1)

Αρνητικά:

- Καταναλώνει χώρο O(V^2)
- Η προσθήκη κόμβου απαιτεί χρόνο O(V^2)

Adjacency List: Ένα array συνδεδεμένων λιστών. Το μέγεθος του array είναι ίσο με τον αριθμό των κόμβων και κάθε εισαγωγή array αντιστοιχεί σε μία συνδεδεμένη λίστα γειτονικών συνδεδεμένων κόμβων [2]

Κάθε index του array αντιστοιχεί σε έναν κόμβο του γράφου



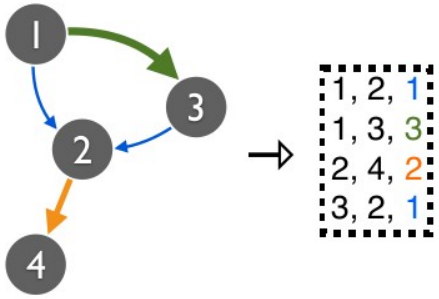
Θετικά:

- Αξιοποιεί χώρο O(|V|+|E|), στη χειρότερη περίπτωση O(V^2)
- Προσθήκη ενός νέου κόμβου είναι ευκολότερη

Αρνητικά:

- Queries που ερευνούν εαν υπάρχει αιχμή από έναν κόμβο σε έναν άλλον δεν είναι αποδοτικά και μπορούν να γίνουν σε χρόνο O(V)

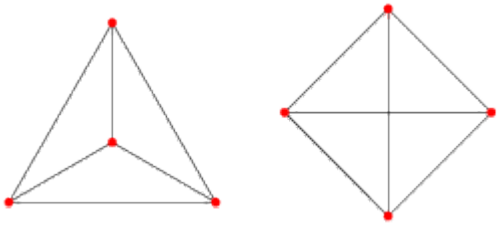
Edge List: Μια από τις πιο συνηθισμένες αναπαραστάσεις γραφημάτων. Μερικές φορές είναι αρκετά δύσκολο να αναλυθεί, ειδικά όταν οι άκρες / κόμβοι έχουν πολλές στήλες. [2]



b) Χρήσιμοι ορισμοί

Επίπεδο γράφημα (planar graph): Ως επίπεδο χαρακτηρίζεται ένα γράφημα του οποίου οι ακμές τέμνονται μόνο σε κορυφές και όχι μεταξύ τους. [1]

Όσο πιο επίπεδο είναι ένα γράφημα τόσο πιο ευδιάκριτες είναι και στην κατανόηση. Υπάρχουν αλγόριθμοι μετατροπής μη επίπεδων γραφημάτων σε επίπεδα και για να διαπιστωθεί αν ένα γράφημα είναι επίπεδο, πρέπει να βρεθεί αν το γράφημα αυτό μπορεί να μετασχηματιστεί σε ένα άλλο ισοδύναμο (ισόμορφο) γράφημα με σαφή απεικόνιση και χωρίς ανεπιθύμητες διασταυρώσεις.



Εικ. 2: Γράφημα σχεδιασμένο σε επίπεδη και σε μη επίπεδη μορφή

Πλήρες γράφημα: Γράφημα στο οποίο όλα τα ζεύγη κορυφών ενώνονται με ακμές του γραφήματος

Ομοιόμορφα γραφήματα: Ζεύγος γραφημάτων στα οποία μπορεί κανείς να περάσει από το ένα στο άλλο προσθέτοντας ή αφαιρώντας κορυφές βαθμού δύο στις ακμές τους.

Ισόμορφα γραφήματα: Ζεύγος γραφημάτων ανάμεσα στα οποία υπάρχει αμφιμονοσήμαντη αντιστοιχία ανάμεσα σε κορυφές και ακμές, και η αντιστοιχία αυτή υπακούει στις σχέσεις γειτνίασης και τους βαθμούς των κορυφών

Υπογράφημα (ενός γραφήματος): Γράφημα που σχηματίζεται από ορισμένες από τις κορυφές ενός άλλου γραφήματος και ορισμένες από τις μεταξύ τους ακμές.

II. Εργαλεία και μέθοδοι

1 Παραλληλοποίηση

Οι εργασίες συνήθως παραλληλίζονται με τρεις διαφορετικούς τρόπους:
Παραλληλισμός διεργασίας: Μια σημαντική διεργασία χωρίζεται σε περισσότερες διεργασίες που μπορούν να λειτουργούν παράλληλα
Παράλληλος δεδομένων: Τα δεδομένα μπορούν να χωριστούν σε πολλά μικρότερα κομμάτια και η διαδικασία μπορεί να εκτελεστεί σε κάθε κομμάτι ανεξάρτητα από τα υπόλοιπα. Τυπικά, αυτές οι τμηματικές διαδικασίες συνδυάζουν στη συνέχεια συγχρονισμένα και μερικώς επεξεργασμένα δεδομένα για να παράγουν το τελικό αποτέλεσμα.

Οι παραπάνω τρόποι παραλληλότητας είναι σχεδόν ανεξάρτητες και είναι δυνατόν να υπάρχουν και τα δύο προβλήματα παραλληλότητας σε ένα προγραμματιστικό μοντέλο.

BSP: Bulk Synchronous Parallelism. Αρχικά δεν είχε φτιαχτεί για επεξεργασία γράφων αλλά για να γεφυρώσει το κενό ανάμεσα σε μοντέλα παράλληλου υπολογισμού και δυνατότητες hardware οι οποίες υποστηρίζουν παραλληλότητα [3].

Υπάρχει ένας διακομιστής ο οποίος χρησιμοποιείται για να μεταφέρει ένα μήνυμα από έναν επεξεργαστή σε έναν άλλο. Την ώρα που ένα ζεύγος κόμβων ανταλλάσει μηνύματα, οποιοσδήποτε τρίτος κόμβος μπορεί να κάνει υπολογισμούς (Barrier Synchronization).

Υπάρχει μία λειτουργία η οποία μπορεί να συγχρονίσει την κατάσταση όλων των αυτόματα υποστατικών διεργασιών. Ο συγχρονισμός αυτός μπορεί να συμβεί περιοδικά σε ένα διάστημα L μονάδων χρόνου είτε βάσει κάποιου διαφορετικού κανόνα. Όποτε αυτό συμβαίνει όμως πρέπει όλοι οι επεξεργαστές που επηρεάζονται να έρθουν σε μία σύμφωνη κατάσταση. Μετά από έναν συγχρονισμό μπορεί να ξεκινήσει ένας καινούριος κύκλος υπολογισμών.

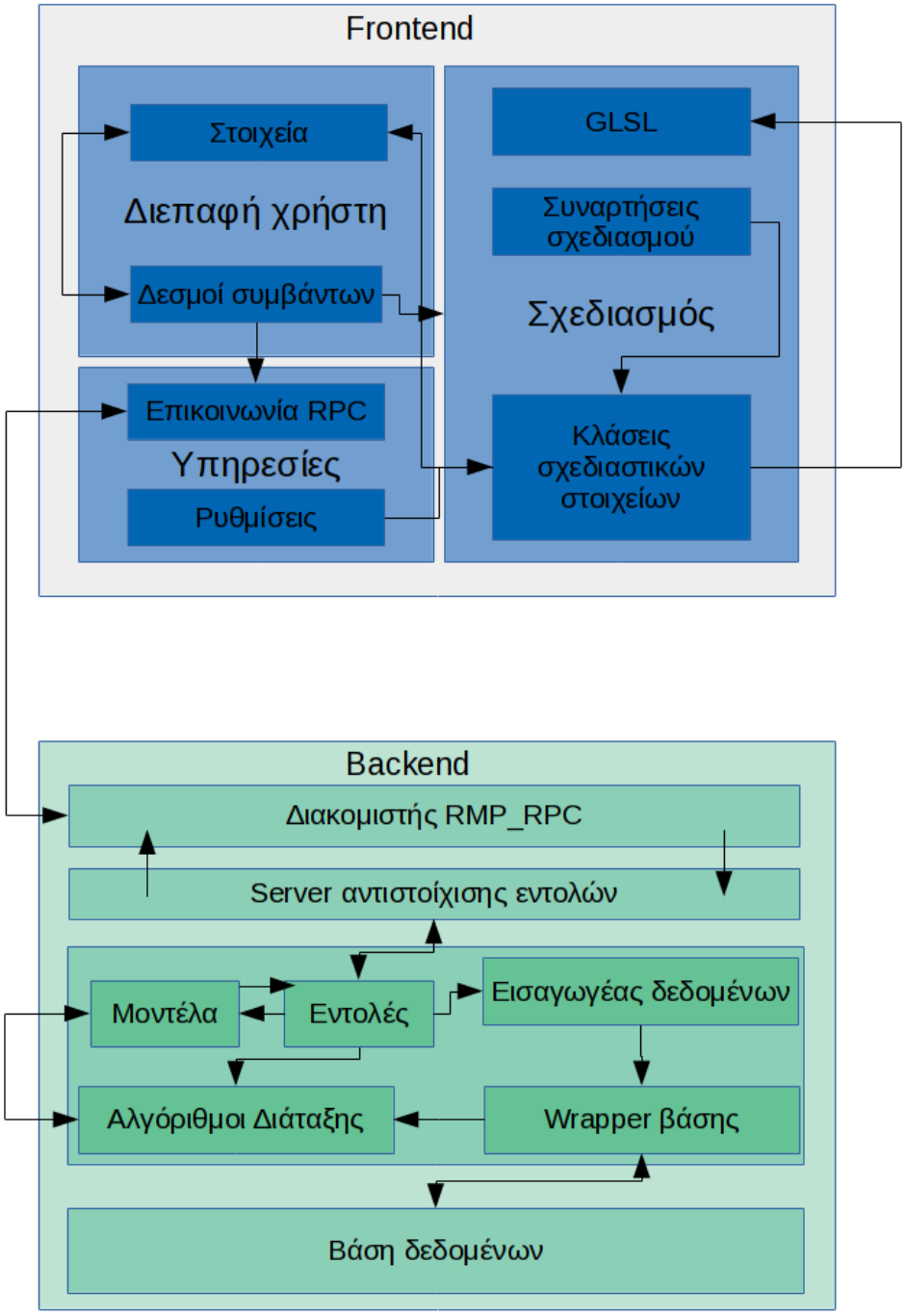
Το παρόν πρόγραμμα παραλληλίζει διεργασίες σχεδιασμού με τη βοήθεια της κάρτας γραφικών, διαμοιράζοντας δηλαδή γραφικά στοιχεία στους πυρήνες της κάρτας γραφικών ή εφαρμόζοντας μάσκες για επιλεκτικό σχεδιασμό ανάλογα με τις ακμές που υπάρχουν μεταξύ κόμβων. Επιπλέον χρησιμοποιεί threads για την ανανέωση του καμβά αλλά και την εφαρμογή κάποιων αλγορίθμων διάταξης. Τέλος υπάρχει σύγχρονη παραλληλότητα μεταξύ της βάσεως δεδομένων και του υπόλοιπου προγράμματος καθώς και εντός του ίδιου μηχανισμού της βάσης με σκοπό την αμεσότερη εφαρμογή queries έτσι ώστε να μην υπάρχει καθυστέρηση σε περιπτώσεις εισαγωγής μεγάλων μεγεθών ή της ανάκλησής τους.

2 Δομή λειτουργίας του προγράμματος

Το πρόγραμμα χωρίστηκε σε frontend και backend με σκοπό την καλύτερη αποσφαλμάτωση με το διαχωρισμό βασικών αρμοδιοτήτων αλλά και την αξιοποίηση κάθε γλώσσας προγραμματισμού για τον αντίστοιχο σκοπό. Το frontend αποτελεί το κομμάτι το οποίο βλέπει ο χρήστης και περιέχει το γραφικό περιβάλλον, το σύστημα σχεδιασμού και διάφορες υπηρεσίες. Λειτουργεί ως client για το backend το οποίο εκτελείται στο παρασκήνιο και παρέχει τη βάση δεδομένων, τον εισαγωγέα δεδομένων από αρχεία αλλά και το σύστημα εφαρμογής των αλγορίθμων όπως και το σύστημα αποθήκευσης της βάσεως δεδομένων.

Η ανάθεση αρμοδιοτήτων έχει γίνει με γνώμονα τον καλύτερο διαμοιρασμό φόρτου εργασίας. Οι δύσκολες εργασίες έχουν ανατεθεί στο backend καθώς αυτό είναι φτιαγμένο σε γλώσσα χαμηλού επιπέδου αλλά και έχει πολύ μεγαλύτερη προστασία από σφάλματα μνήμης το οποίο βοηθάει στην πιο ασφαλή ανταλλαγή πληροφοριών με τη βάση.

Ο σχεδιασμός ανατέθηκε στο frontend καθώς αυτό έχει μία εξαιρετική βιβλιοθήκη που υποστηρίζει hardware acceleration για το σχεδιασμό γραφικών (VisPy).

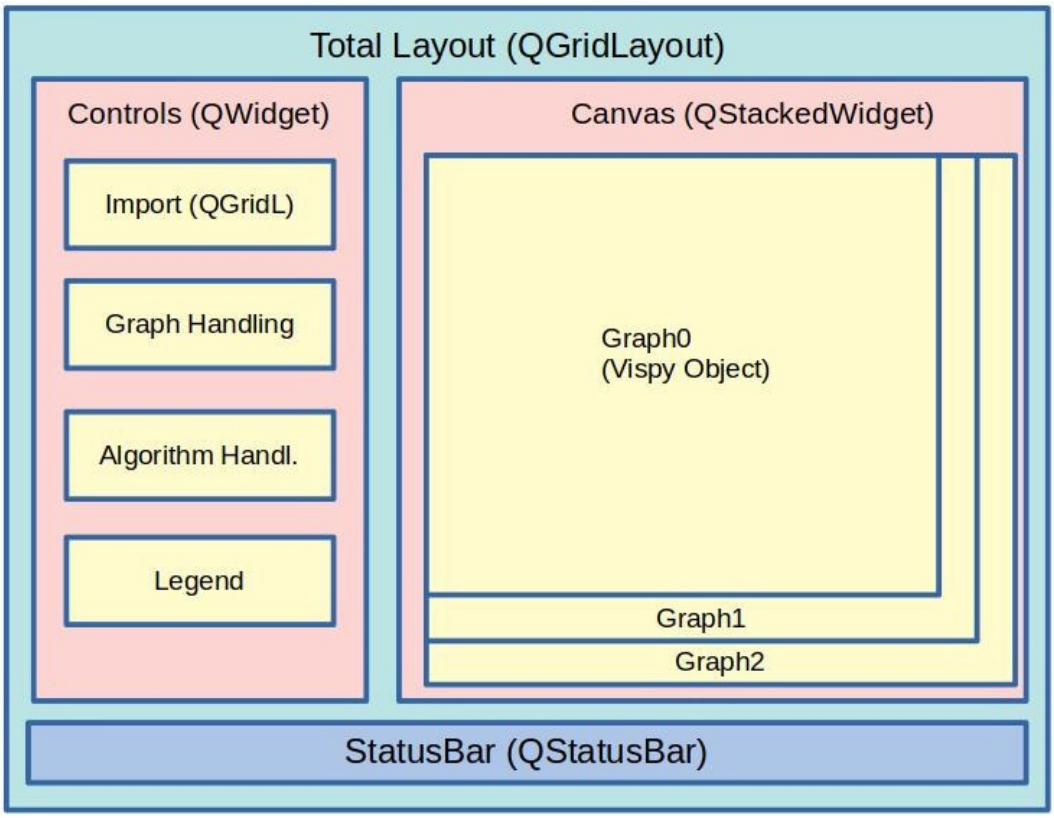


Εικ. 3: Διάγραμμα λειτουργίας του προγράμματος και διαχωρισμός αρμοδιοτήτων

3 To frontend

a) PyQT5

Το PyQt5 είναι η υλοποίηση της βιβλιοθήκης Qt μέσα στη γλώσσα Python [4]. Θεωρείται μια εργαλειοθήκη GUI ανεξάρτητης πλατφόρμας με μια μεγάλη συλλογή στοιχείων (widgets) για να επιλογή. Αυτή η βιβλιοθήκη, μαζί με την αντικειμενοστραφή και γρήγορη υλοποίηση της python3, διευκόλυναν την κατασκευή μιας τέτοιας γραφικής διεπαφής αλλά και την ενσωμάτωση του καμβά προβολής από τη βιβλιοθήκη VisPy στην οποία έχει υλοποιηθεί ο μηχανισμός προεπισκόπησης της οπτικής απεικόνισης των γραφημάτων.
Το χαρακτηριστικό μεγέθους για κάθε κόμβο καθορίστηκε σύμφωνα με τον αριθμό των άκρων που ήταν συνδεδεμένα σε αυτό (σε ένα κανονικό γράφημα) και σύμφωνα με το άθροισμα των βαρών των συνδεδεμένων άκρων (σε περίπτωση βαρυτικού γραφήματος).
Ο χρωματισμός των κόμβων καθορίζει τον τύπο. Μια μέθοδος δημιουργεί μια τυχαία παλέτα χρωμάτων ανάλογα με τον αριθμό των τύπων που υπάρχουν στη συνεδρία γραφήματος.



Εικ. 4: Διάρθρωση γραφικής διεπαφής χρήστη. Οργανώνονται σε διατάξεις και widgets που επικοινωνούν μεταξύ τους με το backend

```
from PyQt5.QtWidgets import (QVBoxLayout, QWidget)
from ..services.actions import Call
from .elements import (AlgorithmControl, GraphControl,
AlgorithmOptions, ImportControl, ColorLegend)


class ControlWidgets(QWidget):
    '''Class that contains the buttons and sliders that control the
graph and general interaction'''

    def __init__(self, parent=None):
        super(ControlWidgets, self).__init__(parent)
        self.selectedCanvasId = 0
        self.algorithm = 'rand uniform'
        self.isSingleFile = False
        self.__controls()
        self.__layout()
        self.__actions()
        self.vbox.addStretch(1)

    def __controls(self):
        self.graphCtrl = GraphControl()
        self.algCtrl = AlgorithmControl()
        self.algOpt = AlgorithmOptions()
        self.importCtrl = ImportControl()
        self.typeList = ColorLegend(self)
        self.typeList.move(60, 200)
        self.typeList.resize(50, 50)

    def __layout(self):
        self.vbox = QVBoxLayout()
        # Add subcomponents
        self.vbox.addLayout(self.importCtrl)
        self.vbox.addLayout(self.graphCtrl)
        self.vbox.addLayout(self.algCtrl)
        self.vbox.addLayout(self.algOpt)
        self.vbox.addWidget(self.typeList)

    def __actions(self):
        # Buttons
        self.algCtrl.algBtn.clicked.connect(self.applyAlg)
        # Dropdowns
        self.algCtrl.algSelector.activated[str]
        .connect(self.algSelect)

    def get_layout(self):
        return self.vbox
...
```

```
 ...
def algSelect(self, text):
        Call.algorithm = text
        print(Call.algorithm)
        if text == 'force directed':
            self.algOpt.enabled('forced', True)
        elif text == 'rand uniform' or text == 'rand normalized':
            self.algOpt.enabled('forced', False)
            self.algOpt.enabled('rand', True)
        else:
            self.algOpt.enabled('forced', False)
            self.algOpt.enabled('rand', False)

    def applyAlg(self):
        self.algCtrl.enableAnimator(False)
        if Call.algorithm == 'force directed':
            algText = self.algOpt.get_text('force directed')
            self.algCtrl.enableAnimator(True)
        elif Call.algorithm == 'rand uniform' or Call.algorithm ==
'rand normalized':
            algText = self.algOpt.get_text('random')
        else:
            algText = ['']
        # Applies distribution algorithm on selected graph
        Call.algParams = algText
        Call.apply_alg()

    # Populate list widget with colors and type names
    def setTypeList(self, colorMap):
        for k in colorMap.keys():
            self.typeList.addType(k, colorMap[k])

    def toggleAnimation(self):
        if self.algCtrl.animateBtn.text == "Animate":
            self.algCtrl.animateBtn.setText("Stop")
            return True
        else:
            self.algCtrl.animateBtn.setText("Animate")
            return False
```

Πλαίσιο 1: Κώδικας Python που αφορά την κεντρική κλάση των controls με τα οποία αλληλεπιδρά ο χρήστης

b) Open Gl

Το Open Gl ES είναι μια διεπαφή προγραμματισμού εφαρμογών (API) για προηγμένα γραφικά 3D που στοχεύουν σε φορητές συσκευές και ενσωματωμένες συσκευές, όπως κινητά τηλέφωνα, PDAs, κονσόλες, συσκευές, οχήματα και αεροναυπηγικά συστήματα. Το Open Gl ES είναι ένα σύνολο API που δημιουργήθηκε από την ομάδα Khronos. Στον κόσμο της επιφάνειας εργασίας υπάρχουν δύο τυπικά API 3D, DirectX και Open Gl. Το Direct X είναι το de facto πρότυπο 3D API για οποιοδήποτε σύστημα που χρησιμοποιεί το λειτουργικό σύστημα Microsoft Windows και χρησιμοποιείται από την πλειοψηφία των παιχνιδιών 3D σε αυτή την πλατφόρμα. Το Open Gl είναι ένα πρότυπο 3D API για πλατφόρμες Linux, διάφορες διανομές UNIX, Mac OS X και Microsoft Windows. Πρόκειται για ένα ευρέως αποδεκτό πρότυπο 3D API που έχει δει σημαντική χρήση στον πραγματικό κόσμο. Το API χρησιμοποιείται από παιχνίδια, διεπαφές χρήστη όπως στο Mac OS X, εφαρμογές CAD (CAD) και οι εφαρμογές δημιουργίας ψηφιακού περιεχομένου. Λόγω της ευρείας υιοθέτησης του Open Gl ως API 3D, ήταν λογικό να αρχίσουμε με το OpenGL API επιφάνειας εργασίας για να αναπτύξουμε ένα ανοικτό πρότυπο 3D API για φορητές και ενσωματωμένες συσκευές και να το τροποποιήσουμε για να καλύψουμε τις ανάγκες και τους περιορισμούς του συγκεκριμένου προγράμματος. Οι περιορισμοί αφορούν το ενδεχόμενο η εφαρμογή να εκτελεστεί σε περιβάλλον χαμηλής απόδοσης αλλά και στο μέλλον σε φορητές συσκευές. [5]

Η βιβλιοθήκη VisPy λειτουργεί ως περιτύλιγμα για την opengl γλώσσα. Με αυτόν τον τρόπο γράφεται ελάχιστος κώδικας GLSL και οι οντότητες που δημιουργούνται σε χαμηλό επίπεδο παραμετροποιούνται, πολλαπλασιάζονται και προβάλλονται βάσει οδηγιών του υπόλοιπου προγράμματος σε αντικειμενοστραφή μορφή. Το γράφημα ως έννοια διασπάστηκε σε τρία τμήματα και δημιουργήθηκαν οι οντότητες κόμβου, αιχμής και βέλους (για τα κατευθυντικά γραφήματα). Αυτές οι οντότητες μετασχηματίζονται με αντικειμενοστρέφεια και τη βοήθεια της βιβλιοθήκης Vispy σε αντικείμενα τα οποία δέχονται παραμέτρους από το υπόλοιπο πρόγραμμα όπως μέγεθος, πλήθος, ποια σημεία ενώνονται μεταξύ τους, χρώματα κλπ.

Στη συνέχεια αυτά εισάγονται σε δύο buffer της κάρτας γραφικών. Το vertex buffer και το fragment buffer έτσι ώστε να λάβει δράση η παραλληλότητα των πυρήνων της κάρτας γραφικών και να γίνει η ταυτόχρονη σχεδίασή τους στον καμβά.

```
#version 120
// Uniforms
// ------------------------------------
uniform float u_antialias;
uniform float u_size;
// 2D scaling factor (zooming).
uniform vec3 u_scale;
uniform vec3 u_pan;
// Attributes
// ------------------------------------
attribute vec3  a_position;
attribute vec4  a_fg_color;
attribute vec4  a_bg_color;
attribute float a_linewidth;
attribute float a_size;
 ...
```

```
...
// Varyings
// ------------------------------------
varying vec4 v_fg_color;
varying vec4 v_bg_color;
varying float v_size;
varying float v_linewidth;
varying float v_antialias;

void main (void) {
    v_size = a_size * u_size;
    v_linewidth = a_linewidth;
    v_antialias = u_antialias;
    v_fg_color  = a_fg_color;
    v_bg_color  = a_bg_color;
    vec3 position_tr = u_scale * (a_position + u_pan);
    gl_Position = vec4(position_tr, 1.0);
    gl_PointSize = v_size*u_scale.x/2 + 2*(v_linewidth +
1.5*v_antialias);
}
```

Πλαίσιο 2: Κώδικας GLSL για τον προγραμματισμό του vertex buffer που αφορά στην οντότητα των κόμβων

```
#version 120

// Constants
// ------------------------------------
// Varyings
// ------------------------------------
varying vec4 v_fg_color;
varying vec4 v_bg_color;
varying float v_size;
varying float v_linewidth;
varying float v_antialias;
// Functions
// ------------------------------------
float marker(vec2 P, float size);
// Main
// ------------------------------------
void main()
{
    float size = v_size +2*(v_linewidth + 1.5*v_antialias);
    // Line width of the markers
    float t = v_linewidth/2.0-v_antialias;
...
```

```
...
    // The marker function needs to be linked with this shader
    float r = marker(gl_PointCoord, size);

    float d = abs(r) - t;
    if( r > (v_linewidth/2.0+v_antialias))
    {
        discard;
    }
    else if( d < 0.0 )
    {
       gl_FragColor = v_fg_color;
    }
    else
    {
        float alpha = d/v_antialias;
        alpha = exp(-alpha*alpha);
        if (r > 0)
            gl_FragColor = vec4(v_fg_color.rgb, alpha*v_fg_color.a);
        else
            gl_FragColor = mix(v_bg_color, v_fg_color, alpha);
    }
}

float marker(vec2 P, float size)
{
    float r = length((P.xy - vec2(0.5,0.5))*size);
    r -= v_size/2;
    return r;
}
```

Πλαίσιο 3: Κώδικας για τον προγραμματισμό του fragment buffer που αφορά στην δημιουργία της οντότητας των κόμβων

c) Δομή κώδικα

Ο κώδικας του frontend του προγράμματος δομήθηκε με μία συγκεκριμένη φιλοσοφία. Στο πρώτο επίπεδο του καταλόγου περιέχεται το πρώτο αρχείο που εκτελείται (main.py) μαζί με το αρχείο παραμετροποίησης (config.yml), το εικονίδιο την άδεια και το readme file. Στον ίδιο φάκελο βρίσκεται και το setup.py το οποίο περιέχει τις ρυθμίσεις σε περίπτωση που χρειαστεί ο κώδικας να γίνει compile έτσι ώστε να είναι αυτόνομος και ανεξάρτητος της ύπαρξης python στο σύστημα.

Στον κατάλογο lib βρίσκεται όλη η βιβλιοθήκη που δημιουργήθηκε για το frontend η οποία είναι διαχωρισμένη στην ομάδα των γραφικών, των υπηρεσιών και των γραφικών στοιχείων. Η ομάδα των γραφικών περιλαμβάνει όλη τη δομή του σχεδιασμού στον καμβά, κάποιες βοηθητικές συναρτήσεις και τον κώδικα της Open GL. Ο κατάλογος των υπηρεσιών περιλαμβάνει την κλήση της διαδικασίας για το backend καθώς και όλες τις εντολές RPC σε στο επίπεδο του client. Επιπλέον περιέχει και τις ρυθμίσεις για την ανάγνωση του αρχείου παραμετροποίησης από το πρόγραμμα. Τέλος, ο κατάλογος των γραφικών στοιχείων έχει διαχωρισμένα τμήματα του γραφικού περιβάλλοντος σε διάφορες κλάσεις με τη μορφή της διάταξής τους στο παράθυρο, τις παραμέτρους τους και τις εντολές που εκτελούν κατά την ενεργοποίησή τους.

```
.
├── main.py
├── assets
│   ├── config.yml
│   └── icon.png
├── lib
│   ├── func.py
│   ├── statics.py
│   ├── graphics
│   │   ├── elements.py
│   │   ├── glsl
│   │   │   ├── a_frag.glsl
│   │   │   ├── a_vert.glsl
│   │   │   ├── e_frag.glsl
│   │   │   ├── e_vert.glsl
│   │   │   ├── n_frag.glsl
│   │   │   ├── n_vert.glsl
│   │   │   └── math
│   │   │       └── constants.glsl
│   │   ├── graph.py
│   │   └── util.py
│   ├── services
│   │   ├── actions.py
│   │   ├── backend.py
│   │   └── config.py
│   └── widgets
│       ├── canvas.py
│       ├── controls.py
│       ├── elements
│       │   ├── algctrl.py
│       │   ├── algoptions.py
│       │   ├── graphctrl.py
│       │   ├── importctrl.py
│       │   └── legend.py
│       ├── statbar.py
│       ├── test.py
│       └── wizard.py
├── LICENSE
└── README.md
```

Πλαίσιο 4: Δομή καταλόγων κώδικα frontend σε Python

4 To Backend

Το backend χωρίστηκε σε τρία τμήματα.

1. Τον κυρίως, που περιέχει τον διακομιστή RPC και το κύριο εκτελέσιμο αρχείο
2. Το lib που περιέχει τις παραμέτρους εντολών RPC, το σύστημα io, το Communicator βάσεων δεδομένων και τις υλοποιήσεις αλγορίθμων
3. Το db-lib που περιέχει τη βιβλιοθήκη βάσεων δεδομένων και το σύστημα αποθήκευσης δεδομένων μνήμης
4. Η συγκεκριμένη γλώσσα είναι συμβατή και με web assembly το οποίο δίνει ακόμα περισσότερες προοπτικές για μελλοντική ανάπτυξη στο cloud ή σε μορφή web application

a) Η γλώσσα προγραμματισμού Rust

Η γλώσσα Rust εφαρμόστηκε από τη Mozilla και είναι εξαιρετική για προγραμματισμό σε επίπεδο συστημάτων. Για τον προγραμματισμό της γραφικής παράστασης του Graphdener χρησιμοποιήθηκε η γλώσσα Rust για τους εξής λόγους:

- Ο νούμερο 1 στόχος αυτής της γλώσσας είναι η ασφάλεια. Εάν ο κώδικας συντάσσεται, εγγυάται ότι δεν θα εκτελεστεί με κανένα σφάλμα. Χαρακτηριστικό παράδειγμα αποτελεί το λάθος των δισεκατομμυρίων δολαρίων σχετικά με τις τιμές Null και τα null references. "Το πρόβλημα με null τιμές είναι ότι αν προσπαθήσετε να χρησιμοποιήσετε μια μηδενική τιμή ως μη-μηδενική τιμή, θα πάρετε κάποιο λάθος. Επειδή αυτή η μηδενική ή μη μηδενική ιδιότητα είναι διαδεδομένη, είναι εξαιρετικά εύκολο να κάνουμε αυτό το είδος σφάλματος " [6]
- Συνδυάζει την ταχύτητα και το χαμηλό επίπεδο της γλώσσας C με την ασφάλεια και την ευκολία γραφής της Python.
- Υποστηρίζει τη διασύνδεση με τη γενική βιβλιοθήκη ArrayFire, η οποία απλοποιεί τη διαδικασία δημιουργίας λογισμικού που στοχεύει παράλληλη και μαζική παράλληλη αρχιτεκτονική, συμπεριλαμβανομένης της CPU, της GPU και άλλων συσκευών
- Έχει ένα εύχρηστο σύστημα αναγνώρισης προτύπων και λειτουργίες που μπορούν να συνδεθούν με τη γλώσσα C

Αρχικά υπήρχε ένα δίλημμα μεταξύ Rust και GO, αλλά επιλέχθηκε η πρώτη επειδή είναι πιο γρήγορη για πειραματικούς / ερευνητικούς λόγους και δεν έχουν γίνει πολλές προσπάθειες να χρησιμοποιηθεί και να αξιολογηθεί για το σκοπό αυτό.

b) Δομή κώδικα

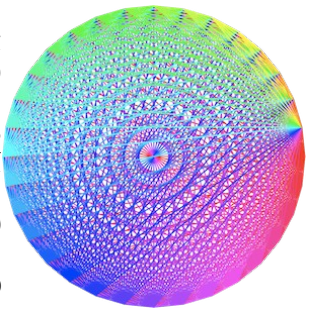
Ο κώδικας του Backend χωρίζεται στους εξής φακέλους. Τον κυρίως που περιέχει μόνο το βασικό αρχείο κλήσης και τον rpc διακομιστή, τη βιβλιοθήκη-κορμό του προγράμματος και τη βιβλιοθήκη της βάσεως δεδομένων. Ο φάκελος που περιέχει τον κορμό του προγράμματος περιέχει τον φάκελο alg ο οποίος αφορά στους μηχανισμούς διατάξεων των γραφημάτων, τον φάκελο commands ο οποίος αφορά στις συναρτήσεις που εκτελούν οι εντολές RPC, το wrapper της βάσης και κλήσεις των αλγορίθμων διάταξης. Επιπλέον ο φάκελος io περιλαμβάνει όλο το σύστημα εισαγωγής δεδομένων από αρχεία και τον επεξεργαστή κοινών

εκφράσεων (regular expressions) ενώ ο φάκελος models περιλαμβάνει τις δομές δεδομένων που μεσολαβούν μεταξύ της αποθήκευσής των στοιχείων ενός γραφήματος στην βάση και του σχεδιασμού του στο frontend.

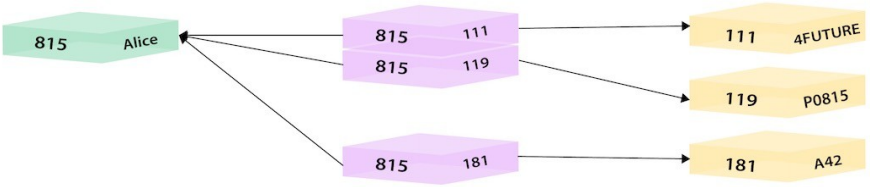
```
.
├── Cargo.lock
├── Cargo.toml
├── graphdener-backend
│   ├── Cargo.lock
│   ├── Cargo.toml
│   └── src
│       ├── main.rs
│       └── server.rs
├── graphdener-lib
│   ├── Cargo.toml
│   └── src
│       ├── alg
│       │   ├── aggregate.rs
│       │   ├── barycenterordering.rs
│       │   ├── circular.rs
│       │   ├── forcedirected.rs
│       │   ├── mod.rs
│       │   └── random.rs
│       ├── commands
│       │   ├── calcs.rs
│       │   ├── database.rs
│       │   ├── distribute.rs
│       │   ├── initials.rs
│       │   ├── mod.rs
│       │   └── retrievals.rs
│       ├── io
│       │   ├── filehandling.rs
│       │   ├── importer.rs
│       │   ├── mod.rs
│       │   └── pattern.rs
│       ├── models
│       │   ├── graph.rs
│       │   ├── mod.rs
│       │   └── nodes.rs
│       ├── lib.rs
│       ├── statics.rs
│       └── traits.rs
└── indradb-lib...
```

Πλαίσιο 5: Δομή καταλόγων κώδικα Backend. Το κυρίως μέρος και η βιβλιοθήκη

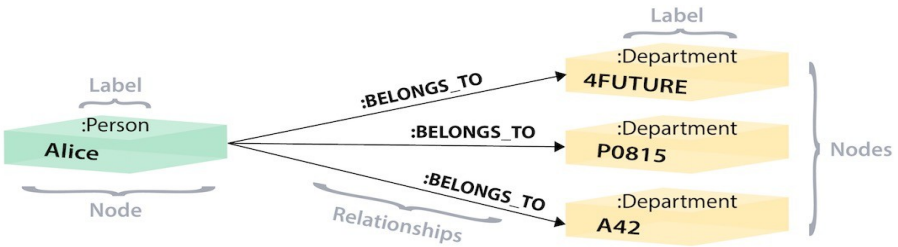
c) Βάση δεδομένων



Για την υλοποίηση της βάσης δεδομένων σε αυτό το έργο, χρησιμοποιήθηκε η βιβλιοθήκη IndraDB [7]. Πρόκειται για ένα σχετικά νέο και πειραματικό σύστημα βάσεων δεδομένων γραμμένο σε Rust. Αποτελείται από το διακομιστή / πελάτη και τη βιβλιοθήκη και υποστηρίζει μεθόδους αποθήκευσης Memory και RocksDB datastore. Για λόγους ταχύτητας και απλότητας, έχει χρησιμοποιηθεί ο τύπος δεδομένων μνήμης και ολόκληρη η βιβλιοθήκη έχει "συσκευαστεί" στο backend σύστημα με τη χρήση ενός αρχείου υπηρεσιών database.rs για να συνδεθεί με το υπόλοιπο πρόγραμμα.
Σε μια παραδοσιακή RDBMS, η καθυστέρηση είναι εκθετική όσο κάθε παράγοντας αυξάνεται ενώ σε μια βάση δεδομένων γραφημάτων η απόδοση παραμένει σταθερή όσο αυξάνεται το γράφημα και η καθυστέρηση παραμένει γραμμική όσο αυξάνεται η πυκνότητα και ο βαθμός.
Μια βάση δεδομένων γραφημάτων δεν έχει καμία σχέση με την οπτικοποίηση αλλά με την επιτυχή εφαρμογή της θεωρίας των γραφημάτων. Χρησιμοποιεί αλγόριθμους που βελτιώνουν το χειρισμό και την αναζήτηση τεράστιων ποσοτήτων δεδομένων. Ένα κύριο χαρακτηριστικό των RDBMS είναι οι περιορισμοί. Αυτό το χαρακτηριστικό τους καθιστά ιδανικό σε περιπτώσεις όπως το κλείδωμα δεδομένων ή η εφαρμογή αυστηρών κριτηρίων και συνθηκών για πολύ σταθερά και ειδικά συστήματα. Αντίθετα, όταν πρόκειται για δεδομένα από τον πραγματικό, παρατηρήσιμο κόσμο, οι σχέσεις μεταξύ αντικειμένων και της εμφάνισής τους μπορεί να γίνουν δύσκολες έως αδύνατες με τη χρήση ενός RDBMS. Αυτό το πρόβλημα αντιμετωπίζεται επιτυχώς από βάσεις δεδομένων με γραφικούς κόμβους και αιχμές, όπου οι κόμβοι έχουν ιδιότητες {key: value} αλλά και "ετικέτες" που βοηθούν την κατηγοριοποίησή τους (π.χ. πρόσωπο, μόριο, χρήστης, αντικείμενο) την κατεύθυνση καθώς και τις ιδιότητες.



Εικ. 5: Σχεσιακή βάση - πίνακες προσώπων και διδακτικών μονάδων [13]



Εικ. 6: Βάση γραφήματος – Το άτομο Alice και 3 Τμήματα ως κόμβοι [13]

Ο κώδικας της βιβλιοθήκης της βάσεως δεδομένων χωρίζεται σε δύο φακέλους τύπων

αποθήκευσης και έναν φάκελο μοντέλων. Επιπλέον περιέχονται και κάποια αρχεία που αφορούν τα σφάλματα, την απόδοση μοναδικών κλειδιών (UUIDs) και traits (αντίστοιχη φιλοσοφία με τα interfaces σε C# και Java). Οι φάκελοι τύπων αποθήκευσης είναι μνήμης και RocksDB (μία βάση δεδομένων που έχει δημιουργήσει το Facebook για να εξυπηρετεί τις ανάγκες του). Ο κατάλογος των μοντέλων περιέχει τις δομές που απαρτίζουν τα στοιχεία ενός γραφήματος έτσι ώστε να υπάρχει λογική γύρω από τον τρόπο αποθήκευσής τους και ανάκλησής τους. Παράδειγμα αποτελεί το αρχείο edge.rs και vertices.rs που αφορούν τις αιχμές και τους κόμβους αντίστοιχα.

```
├──indradb-lib
   ├── Cargo.toml
   └── src
       ├── benches
       │   ├── benches.rs
       │   ├── macros.rs
       │   └── mod.rs
       ├── memory
       │   ├── datastore.rs
       │   └── mod.rs
       ├── models
       │   ├── edges.rs
       │   ├── metadata.rs
       │   ├── mod.rs
       │   ├── queries.rs
       │   ├── types.rs
       │   └── vertices.rs
       ├── rdb
       │   ├── datastore.rs
       │   ├── instance.rs
       │   ├── keys.rs
       │   ├── managers.rs
       │   └── mod.rs
       ├── errors.rs
       ├── lib.rs
       ├── traits.rs
       └── util.rs
```

Πλαίσιο 6: Δομή καταλόγων κώδικα backend. Η βιβλιοθήκη της βάσης δεδομένων

d) c) ArrayFire

Το ArrayFire είναι μια βιβλιοθήκη λογισμικού υψηλής απόδοσης για παράλληλες υπολογιστικές εφαρμογές με ένα εύκολο στη χρήση API. Το σύστημά του είναι βασισμένο σε arrays που καθιστούν τον προγραμματισμό πιο προσβάσιμο και ταχύτερο για μεγάλες ποσότητες δεδομένων. Επιπλέον υποστηρίζεται από πολλές γλώσσες συμπεριλαμβανομένης και της γλώσσας Rust. Στο πρόγραμμα χρησιμοποιείται το API της ArrayFire σε περιπτώσεις όπως η τυχαία διάταξη έτσι ώστε να εκμηδενιστεί ο χρόνος υπολογισμού των τυχαίων σημείων ενός εξαιρετικά μεγάλου γραφήματος.

e) Εισαγωγέας δεδομένων

Το πρόγραμμα διαθέτει έναν εισαγωγέα δεδομένων (importer) ο οποίος βασίζεται σε regular expressions για να εξορύσσει τα δεδομένα απο της στήλες ενός αρχείου το οποίο μπορεί να περιέχει πληροφορίες για το γράφημα. Τα δεδομένα γραφημάτων μπορούν να έχουν διάφορες μορφές. Μερικές φορές υπάρχει απλώς ένα μοναδικό αρχείο το οποίο περιέχει πληροφορίες για τις αιχμές (κόμβος αρχικός, κόμβος τελικός) το οποίο ουσιαστικά πρόκειται για ένα μεγάλο edge list. Μπορεί επίσης να περιέχει επιπρόσθετες πληροφορίες όπως βάρος αιχμής, ταυτότητα ή τύπος. Ο δέυτερος τρόπος αποθήκευσης είναι σε διαχωρισμένα αρχεία (συνήθως δύο) τα οποία περιέχουν πληροφορίες για τους κόμβους και τις αιχμές αντίστοιχα. Αυτός ο τρόπος συνήθως παρέχει πιο αναλυτικές πληροφορίες και για τους κόμβους και για τις αιχμές.Στο πρόγραμμα ο εισαγωγέας δεδομένων έχει διαχωριστεί σε δύο τμήματα.

Το πρώτο, αποτελεί μέρος της διεπαφής χρήστη και βρίσκεται υπό τη μορφή ενός wizard όπου ο χρήστης μπορεί σε δύο βήματα να εισάγει ένα ή δύο αρχεία δεδομένων. Το πρώτο βήμα αφορά τις πληροφορίες των κόμβων ενώ το δεύτερο τις πληροφορίες των αιχμών. Στην περίπτωση που οι πληροφορίες βρίσκονται σε ένα και μόνο αρχείο, το ίδιο πρέπει να εισαχθεί και στα δύο βήματα. Στο πρώτο βήμα ως στήλες ID κόμβων (οποιεσδήποτε στήλες περιέχουν ID κόμβων πρέπει να ορισθούν), ενώ στο δεύτερο βήμα ως στήλες κόμβου έναρξης και κόμβου άφιξης για να δημιουργηθούν οι αιχμές.

Ο τρόπος με τον οποίο ο χρήστης ενημερώνει το πρόγραμμα για τη θέση των πληροφοριών αλλά και τη μορφή με την οποία είναι διαχωρισμένες γίνεται με τη χρήση πεδίων κειμένου και επιλογέων. Κάθε επιλογέας αντιπροσωπεύει το όνομα μίας στήλης από το αρχείο. Εαν η στήλη δεν αποτελεί ενδιαφέρον ο χρήστης πρέπει να επιλέξει την παύλα (-), διαφορετικά επιλέγει ένα απο τα ονόματα των υποστηριζόμενων τύπων πληροφορίας (ID, type, label, weight, from, to,

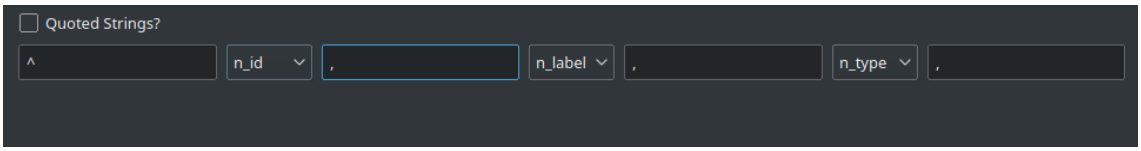
```
24 1
595 1
1143 1
1392 1
1405 1
156 2
951 2
1033 2
459 3
997 3
595 4
...
```

Πλαίσιο 7: Ενιαίο αρχείο αιχμών και κόμβων. Περιέχει μόνο ID κόμβου προέλευσης και ID κόμβου προορισμού

Στην περίπτωση που η πληροφορία βρίσκεται χωρισμένη σε αρχείο κόμβων και αρχείο αιχμών, Ακολουθείται η ίδια διαδικασία απλά σε κάθε ένα από τα βήματα επιλέγεται το αντίστοιχο αρχείο από τον περιηγητή.

```
1,Robert C. Allen,person,individual,0,0.1708217782071894,0,2,2,2
2,Philip Ethington,person,individual,0,0.11525159594332783,0,2,2,2
3,Chien-Yi Hou,person,individual,0,0.1708217782071894,0,2,2,2
4,Christopher
Johanson,person,individual,0,0.16513694022027706,0,3,3,3
5,Pamella Lach,person,individual,0,0.1708217782071894,0,2,2,2
...
```

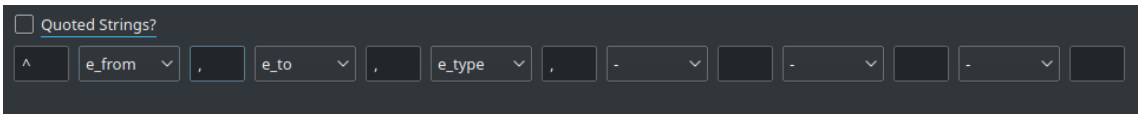
Πλαίσιο 8: Δείγμα μίας εισαγωγής δεδομένων από ξεχωριστά αρχεία. Το συγκεκριμένο περιέχει πληροφορίες κόμβων [14]



Εικ. 7: Wizard εισαγωγής βήμα 1ο. Οι παράμετροι που ταιριάζουν με τη μορφοποίηση του παραπάνω δείγματος εισαγωγής κόμβων από αρχείο. Η έναρξη ανάγνωσης συμβολίζεται με το ^ ενώ οι διαχωριστές είναι με κόμματα χωρίς κενά

```
1,353,Directed,1,,1.0
1,1416,Directed,1057,,1.0
2,353,Directed,2,,1.0
2,1524,Directed,1038,,1.0
3,353,Directed,3,,1.0
3,1416,Directed,826,,1.0
4,353,Directed,4,,1.0
4,485,Directed,359,,1.0
4,1495,Directed,833,,1.0
5,353,Directed,5,,1.0
5,1416,Directed,1031,,1.0
...
```

Πλαίσιο 9: Δείγμα μίας εισαγωγής δεδομένων από ξεχωριστά αρχεία. Το συγκεκριμένο περιέχει πληροφορίες αιχμών [14]



Εικ. 8: Wizard εισαγωγής δεδομένων από αρχείο αιχμών. Παρατηρείται ότι μόνο οι τρεις πρώτες στήλες αποτελούν ενδιαφέρον. Οι υπόλοιπες ορίζονται ως κενές με τη χρήση της παύλας στον επιλογέα στηλών

III. Αλγόριθμοι διάταξης

Ο εύμορφος σχεδιασμός γραφημάτων αφορά κάποιο είδος της ιεράρχησης ενός συνόλου αισθητικών κριτηρίων.

Δεν υπάρχει κανένας καθολικός αλγόριθμος ο οποίος παράγει όμορφα σχέδια για κάθε είδος γραφικής εφαρμογής. Επομένως υπάρχουν πολλοί αλγόριθμοι στη βιβλιογραφία που προσπαθούν να παράγουν καλή αυτόματη διάταξη για κάθε οικογένεια γραφημάτων.

Υπάρχουν αρκετά έργα τα οποία σχετίζονται με τη γενική διάταξη γραφημάτων και είναι δυνατή η ταξινόμηση αυτών των αλγορίθμων κάτω από διάφορους τίτλους. Ωστόσο, μόνο οι δύο τύποι διατάξεων που περιγράφονται παρακάτω σχετίζονται με αυτή τη μελέτη.

1 Δυναμικά κατευθυνόμενη διάταξη γραφήματος (force-directed)

Ο υπολογισμός των ελκτικών δυνάμεων είναι αποδοτικός,
απαιτώντας χρόνο O(|E|). Ωστόσο, οι απωστικές δυνάμεις απαιτούν κάθε ζεύγος κορυφών να συγκρίνονται σε χρόνο $O(|V|^2)$. Για να βελτιωθεί η απόδοση του δεύτερου υπολογισμού, οι απωστικές δυνάμεις που υπολογίζονται για μια κορυφή u είναι περιορισμένες σε όσες κορυφές βρίσκονται μέσα σε μία ακτίνα k = √ύψος x πλάτος. Ως λύση έχει προταθεί να διαχωριστεί ο χώρος σχεδίασης σε ένα πλέγμα με κελιά μεγέθους k × k όπου οι κορυφές ταξινομούνται σε αυτά βάσει των θέσεών τους.[8]

Στην πράξη, το γράφημα που απεικονίζεται παρουσιάζεται ως φυσικό μοντέλο που διέπεται από βασικούς νόμους της φύσης και η προσομοίωση αυτού του μοντέλου γίνεται με μια προσέγγιση. Δηλαδή το πρόβλημα της διάταξης επιλύεται μέσω της προσομοίωσης ενός φυσικού συστήματος. Στο βασικό μοντέλο, οι κόμβοι συμπεριφέρονται ως φορτισμένα σωματίδια που απωθούν το καθένα ενώ οι αιχμές παριστάνονται με ελατήρια. Το επίπεδο ενέργειας ενός κόμβου καθορίζεται από τις δυνάμεις που ενεργούν σε αυτό. Το ελατήριο προσπαθεί να ελαχιστοποιήσει το συνολικό επίπεδο ενέργειας μετακινώντας τους κόμβους προς την κατεύθυνση των δυνάμεων. [9]

Το ολικό επίπεδο ενέργειας, το οποίο είναι το άθροισμα όλων των ενεργειακών επιπέδων των κόμβων, υπολογίζεται μετά από κάθε επανάληψη για να καθορίσει εάν η συνολική ενέργεια είναι κάτω από ένα συγκεκριμένο ποσοστό. Η ακρίβεια και η πραγματικότητα αυτού του βασικού συστήματος είναι ανάλογες της ισορροπίας μεταξύ απόδοσης και ποιότητας. Γενικά, για λόγους απόδοσης μόνο ένας κόμβος μετατοπίζεται ανά επανάληψη.[q] Ωστόσο στο συγκεκριμένο έργο γίνονται πρώτα οι υπολογισμοί των δυνάμεων σε βάθος χρόνου dt και στη συνέχεια ενημερώνονται οι θέσεις των κόμβων για να επανασχεδιαστούν στο frontend. Είναι πάντα δυνατό να συμπεριληφθούν πρόσθετοι φυσικοί παράγοντες στο μοντέλο με σκοπό μία πιο ρεαλιστική συμπεριφορά. Μερικά παραδείγματα επιπρόσθετων παραμέτρων είναι τα παρακάτω:

1. Μαγνητικές δυνάμεις: Αυτές χρησιμοποιούνται γενικά για την επιβολή μιας ροής στο σχεδιασμό. Για παράδειγμα σε κατευθυντικά γραφήματα είναι δυνατή η έμφαση στη ροή εάν όλα τα άκρα ερμηνεύονται ως πυξίδες οι οποίες ευθυγραμμίζονται σύμφωνα με κάποιο μαγνητικό πεδίο.
2. Βαρυτικές δυνάμεις: Χρησιμοποιούνται γενικά για την παραγωγή πιο συμπαγών σχεδιασμών. Καθώς οι δυνάμεις του ελατηρίου είναι αποτελεσματικές μόνο μεταξύ των συνιστωσών και οι απωστικές δυνάμεις μπορούν να κάνουν την έκταση που καταλαμβάνεται μόνο μεγαλύτερη. Το βασικό μοντέλο πρέπει να επεκταθεί ώστε να είναι σε θέση να ελαχιστοποιήσει το διάστημα μεταξύ των συνιστωσών Ως εκ τούτου, εισάγονται βαρυτικές συνιστώσες. Όλοι οι κόμβοι προσελκύονται από το κέντρο μάζας όλων των υπόλοιπων κόμβων.
3. Επιτάχυνση: Είναι επίσης δυνατό να προστεθεί η ορμή συντελεστή που σχετίζεται με τη μάζα του μοντέλου. Αυτοί οι παράγοντες προσθέτουν την προηγούμενη ταχύτητα

ενός κόμβου επί της κίνησής του η οποία υπολογίζεται για μία επανάληψη. Η επιτάχυνση γενικά προστίθεται για να βελτιώσει την απόδοση του χρόνου εκτέλεσης καθώς και την ποιότητα.

4. Θερμοκρασία: Το βασικό μοντέλο με τις επεκτάσεις του που περιγράφονται μέχρι τώρα να μπορεί να έρθει σε ένα τοπικό ελάχιστο. Για να ξεπεραστεί αυτό το πρόβλημα είναι δυνατό να χρησιμοποιηθεί μία ελεγχόμενη ποσότητα τυχαιότητας. Για να γίνει αυτό αξιοποιείται η τεχνική της προσομοίωσης ανόπτησης που επιτρέπει αλλαγές σε καταστάσεις υψηλότερης ενέργειας. Με την προσθήκη αυτή η υπολογισμένη κίνηση ενός κόμβου διαταράσσεται από ένα σχετικά μικρό τυχαίο διάνυσμα για να αποφευχθεί η παγίδευσή του σε κάποιο τοπικό ενεργειακό ελάχιστο. Αρχικά το μέγεθος αυτού του τυχαίου διανύσματος είναι μεγαλύτερο, ενώ καθώς η προσομοίωση εξελίσσεται το σύστημα "ψύχεται" που σημαίνει ότι το μέγεθος της δύναμης μειώνεται προκειμένου να σταθεροποιηθεί σε μία τελική διάταξη [10].

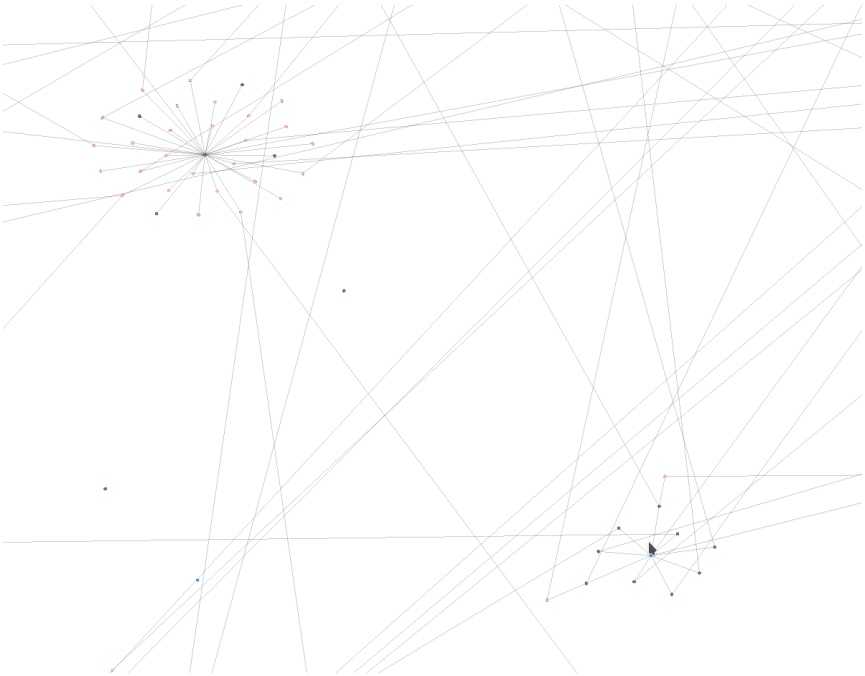
Ο συγκεκριμένος αλγόριθμος εκτελείται βάσει των βασικών τεσσάρων παραμέτρων (σταθερά ελατηρίου, απωστική δύναμη, μήκος ανάπαυσης ελατηρίου, χρονικό βήμα) σε επαναλήψεις ανά καρέ σχεδιασμού.

```
const MAX_DISPLACEMENT_SQUARED: f32 = 56.0;
// L = spring rest length
// K_r = repulsive force constant
// K_s = spring constant
// delta_t = time step
pub fn force_directed(graph: &mut Graph, l: f32, k_r: f32, k_s: f32,
deltat: f32) → () {
    let (tx, rx) = mpsc::channel();
    let mut nodes_r = graph.nodes.clone();
    let mut nodes_s = graph.nodes.clone();
    let tx1 = mpsc::Sender::clone(&tx);
    // repulsion between all pairs
    thread::spawn(move || {
        let nodes = repulsion(nodes_r, k_r);
        for node in nodes {
            tx1.send(node).unwrap();
        }
    });
    thread::spawn(move || {
        // spring force between adjacent pairs
        let nodes = spring(nodes_s, k_s, l);

        for node in nodes {
            tx.send(node).unwrap();
        }
    });
    for received in rx {
        // update positions
        update(received, deltat, graph);
    }
}
```

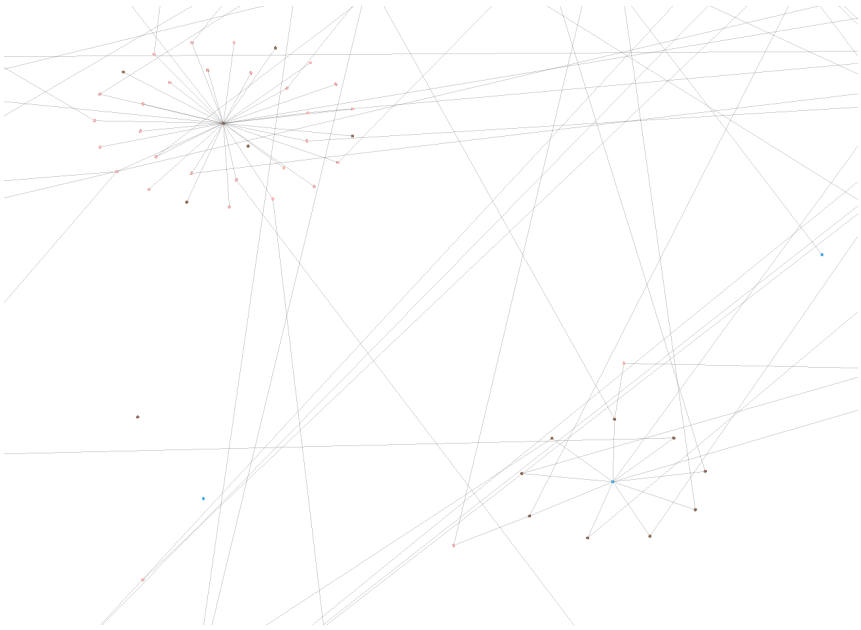
Πλαίσιο 10: Μέρος κώδικα δυναμικά κατευθυνόμενης διάταξης σε Rust

a) Spring Rest Length

This value determines the length of the stretch each edge can withstand.



Εικ. 9: Μήκος ανάπαυσης ελατηρίου = 10



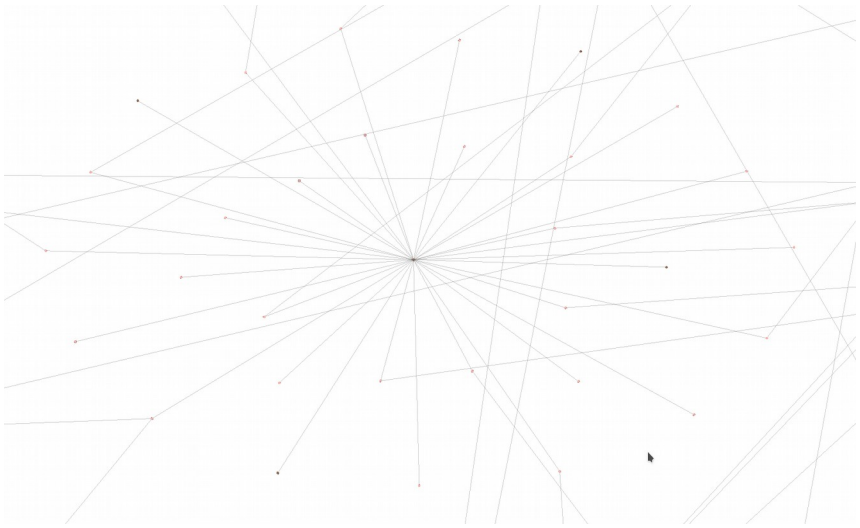
Εικ. 10: Μήκος ανάπαυσης ελατηρίου = 40

b) Σταθερά απωστικής δύναμης (Repulsive force constant)

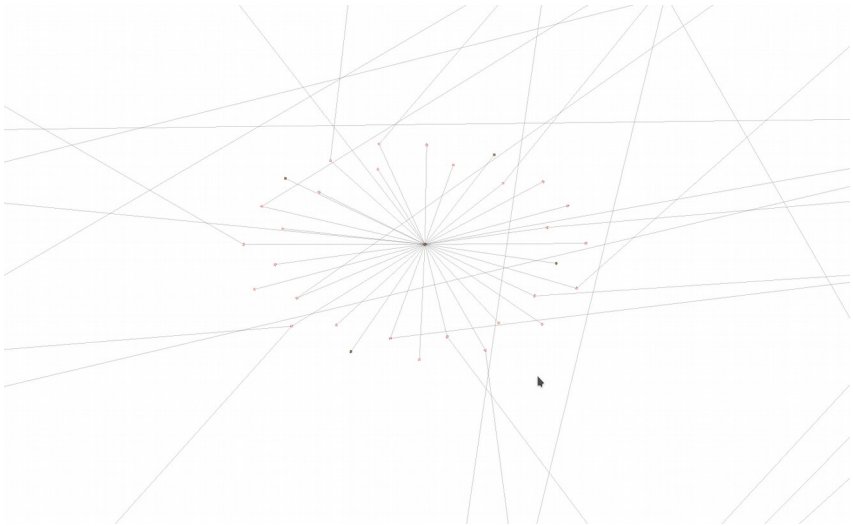
Η συγκεκριμένη σταθερά ορίζει το πόσο έντονη είναι η δύναμη μεταξύ των κόμβων. Η τιμή αυτή μπορεί να είναι και αρνητική. Γενικά βοηθάει να χρησιμοποιηθούν διαφορετικές τιμές για το χρόνο που τρέχει ο αλγόριθμος έτσι ώστε να έρθουν σε μία επιθυμητή μορφή γραφήματα τα οποία είναι αρκετά περίπλοκα και έχουν υψηλό βαθμό συνδέσεων.

c) Σταθερά ελατηρίου (Spring constant)

Αυτή η σταθερά αντιπροσωπεύει την "ακαμψία" των άκρων. Όσο χαμηλότερος είναι ο αριθμός, τόσο πιο δύσκολο είναι να οργανωθούν οι κόμβοι γύρω από άλλους κόμβους. Όσο υψηλότερος είναι ο αριθμός, τόσο πιο χαοτική είναι η κατανομή των κλικών με έναν κεντρικό κόμβο.



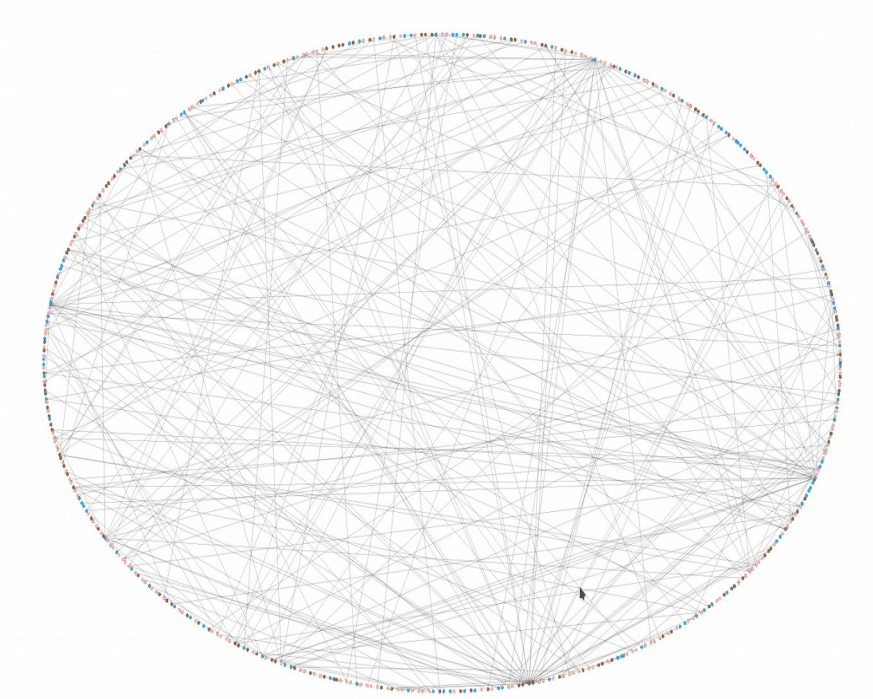
Εικ. 11: Υπογράφημα με σταθερά ελατηρίου = 0.06



Εικ. 12: Υπογράφημα με σταθερά ελατηρίου = 6

2 Περιμετρικός (Circular)

Αποτελεί μία αρκετά “καθαρή” λύση αλλά και μία αρχική κατανομή με σκοπό την εφαρμογή ενός δεύτερου αλγορίθμου σε αυτήν μετέπειτα, όπως ο δυναμικά κατευθυνόμενος.



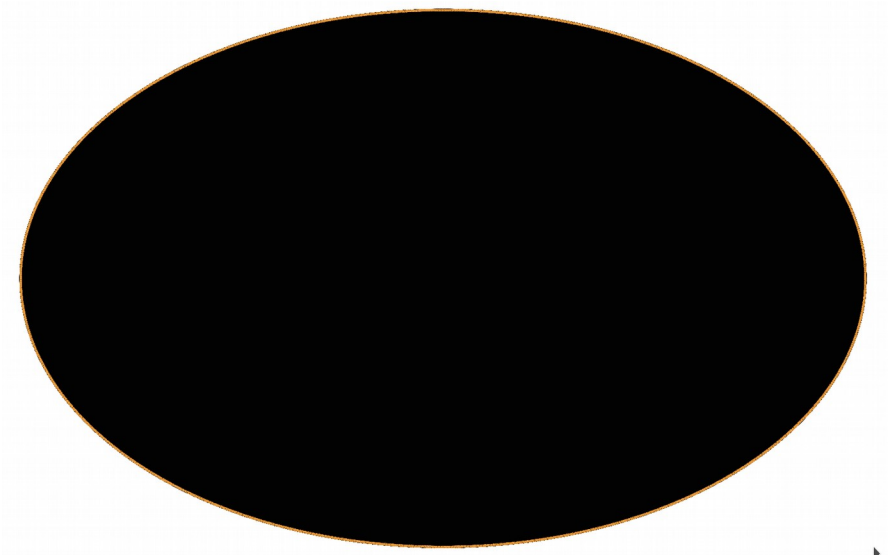
Εικ. 13: Γεωμετρικός (κυκλικός) αλγόριθμος αναπαράστασης

Όταν πρόκειται για εξαιρετικά μεγάλα γραφήματα είναι πρακτικό να γίνει εκκίνηση πρώτα με τον περιμετρικό αλγόριθμο και στη συνέχεια να εφαρμοστεί ο δυναμικά κατευθυνόμενος αλγόριθμος. Παρακάτω δίνεται παράδειγμα ενός ιδιαίτερα μεγάλου γραφήματος με μετάβαση από περιμετρική διάταξη σε δυναμικά κατευθυνόμενη μέσα σε βάθος 10 λεπτών.

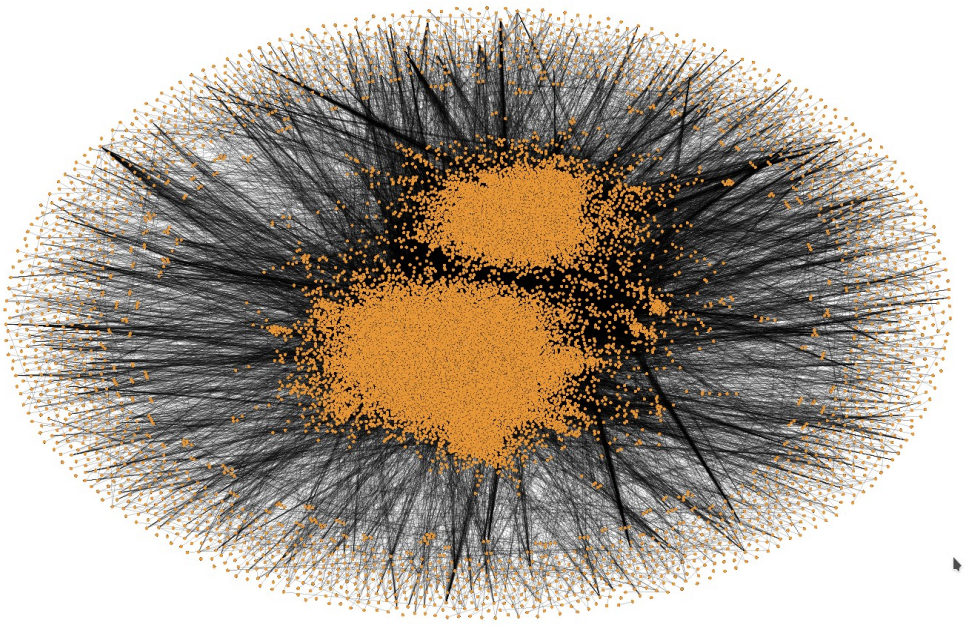
Σε αυτή την έκδοση του αλγορίθμου δεν εφαρμόζεται επιπλέον διαδικασία μετατροπής του γραφήματος σε ένα προσεγγιστικά επίπεδο γράφημα διότι θα απαιτούνταν ιδιαίτερα μεγάλη υπολογιστική ισχύς και χρόνος.

```
fn r_from_circumference(circumference: f64) → f64 {
    circumference / (2. * PI)
}

fn polar_to_rect(theta: f64, radius: f64) → (f64, f64) {
    let y = radius * theta.sin();
    let x = radius * theta.cos();
    (x, y)
}

pub fn polygon(n: u32) → Vec<(f64, f64)> {
    let point_size: f64 = 1.;
    let circumference = point_size * (n as f64) / PI;
    let sides = n;
    // Find the angle between corners in degrees
    let degrees = 360. / (sides as f64);
    let r = r_from_circumference(circumference);
    let mut result: Vec<(f64, f64)> = Vec::new();
    let mut theta = 0.;

    while theta < 360. {
        let point = polar_to_rect(theta, r);
        result.append(&mut vec![point]);
        theta = theta + degrees;
    }
    result
}
```

Text 1: Κώδικας αλγορίθμου περιμετρικής διάταξης σε Rust



Εικ. 14: Γράφημα 314795 κόμβων και 23180 αιχμών σε περιμετρική διάταξη με επιστημονικές παραθέσεις [15]



Εικ. 15: Το γράφημα της εικ. 13 έπειτα από μετάβαση σε δυναμικά κατευθυνόμενη διάταξη σε βάθος 8 λεπτών

3 Τυχαία διάταξη (random)

Προκειμένου να αποφευχθούν προβλήματα επιδόσεων, ειδικά όταν εμπλέκονται γραφήματα με πάνω από 100000 άκρα , χρησιμοποιήθηκε η τυχαία αριθμοδότηση της σουίτας arrayfire έτσι ώστε να αξιοποιηθεί η λειτουργικότητα της GPU. Οι λειτουργίες randu και randn χρησιμοποιήθηκαν για να επιλεχθούν δύο διαφορετικές διατάξεις τυχαιότητας. Συνήθως αυτός ο αλγόριθμος χρησιμοποιείται ως προετοιμασία πριν από την εφαρμογή ενός πιο οργανωμένου.

Στο πρόγραμμα υπάρχουν δύο τύποι τυχαίων κατανομών. Η κανονική και η ομοιόμορφη. Η πρώτη χαρακτηρίζεται από την πυκνή συγκέντρωση κόμβων στο κέντρο η οποία αραιώνει προς τα έξω ενώ η δεύτερη καταλαμβάνει έναν σταθερό χώρο με διαμοιρασμένους κόμβους με ομοιόμορφη πυκνότητα σε όλο το επίπεδο.

Η τυχαία κατανομή και ιδιαίτερα η κανονική υποκατηγορία της, είναι ιδανική σε περίπτωση που υπάρχουν πολλοί ορφανοί κόμβοι (κόμβοι μηδενικού βαθμού) οι οποίοι μετά την εφαρμογή του δυναμικά κατευθυνόμενου αλγόριθμου θα απομακρυνθούν ευκολότερα.

a) Scatter factor

Ο παράγοντας σκέδασης είναι η μόνη παράμετρος για αυτό το μοντέλο. Αυξάνει την απόσταση μεταξύ κάθε τυχαία τοποθετημένου κόμβου και η αύξηση του είναι λογαριθμική. Ο παράγοντας που εισάγεται από τον χρήστη πολλαπλασιάζεται με το ln (n) όπου n είναι ο συνολικός αριθμός κόμβων. Αυξάνοντας έτσι την εξάπλωση και σύμφωνα με το μέγεθος του γραφήματος.

IV. Παραμετροποίηση

Η εφαρμογή διαθέτει ένα αρχείο config.yml το οποίο περιέχει πολλές διαφορετικές παραμέτρους που μπορούν να προσαρμόσουν τη συμπεριφορά του συστήματος προβολής, όπως το μέγεθος του δείκτη, ορισμένες σταθερές, τα ονόματα των στηλών κ.λπ.

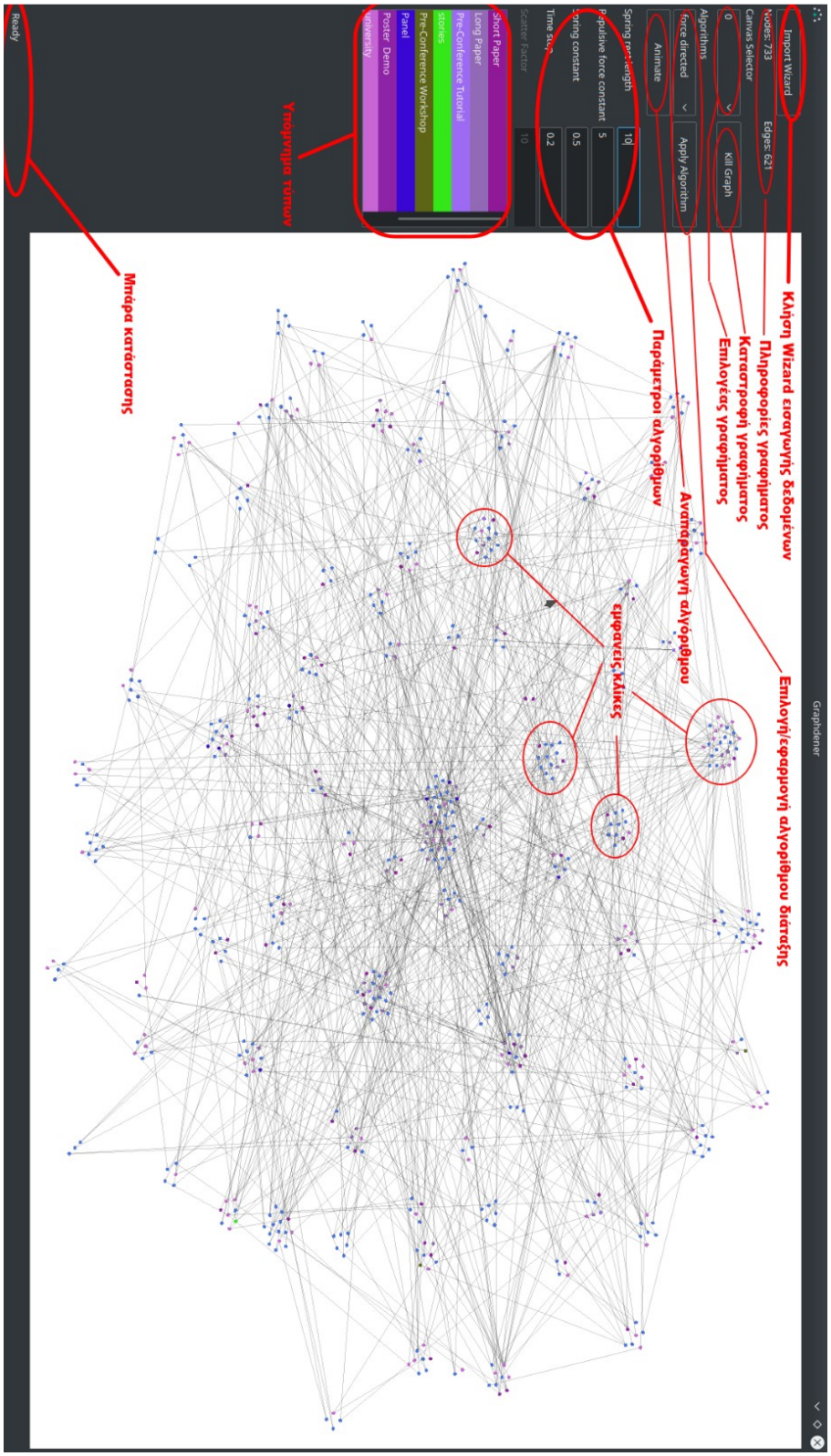
Υπάρχει η δυνατότητα μέσω του configuration να ορισθούν νέα ονόματα πληροφοριών. Η πληροφορία χωρίζεται σε απαραίτητη και προαιρετική. Απαραίτητη πληροφορία είναι αυτή η οποία είναι αναγκαία για το σχεδιασμό γραφημάτων και είναι το ID, ο τύπος, η προέλευση και ο προορισμός. Οτιδήποτε άλλο προστεθεί από το χρήστη όπως το label και το βάρος, αποτελεί προαιρετική πληροφορία και αποθηκεύεται με τη μορφή μεταδεδομένων (metadata) στη βάση δεδομένων.

Επιπλέον ο χρήστης οφείλει να ορίσει σύμβολα τα οποία μπορεί να περιέχονται σε αλφαριθμητικές στήλες όπως το label και type τα οποία δεν θέλουμε να γίνουν αντιληπτά ως ειδικοί χαρακτήρες. Π.χ. μπορεί να υπάρχει στήλη με label “Evaluating Digital Scholarship: A Case Study in the Field of Literature” και η μορφοποίηση του αρχείου να μην ξεχωρίζει τα αλφαριθμητικά με εισαγωγικά. Σε αυτή την περίπτωση εαν ο χρήστης δεν έχει εξαιρέσει την άνω και κάτω τελεία μπορεί να δημιουργηθεί πρόβλημα στον εισαγωγέα.

```
directories:
  backend:
    - /home/orestes/Workspace/src/github.com/moreorem/graphdener-
backend/target/debug/
constants:
  alg:
    - 'random'
    - 'circular'
    - 'force directed'
  forcelabels:
    - 'Spring rest length'
    - 'Repulsive force constant'
    - 'Spring constant'
    - 'Time step'
  nodecnames:
    - 'n_id'
    - 'n_label'
    - 'n_type'
  edgecnames:
    - 'e_id'
    - 'e_from'
    - 'e_to'
    - 'e_label'
    - 'e_type'
    - 'e_weight'
  columntypes:
    {
      'n_id': 'int',
      'n_label': 'str',
      'n_type': 'str',
      'e_weight': 'int',
      'e_from': 'int',
      'e_to': 'int',
      '_': '',
      'e_id': 'int',
      'e_label': 'str',
      'e_type': 'str'
    }
graph:
  marker_size: 40
  arrow_size: 0.05
  framecap: 60
importer:
  acceptedsymbols: :.()&'-
```

Πλαίσιο 11: Το αρχείο παραμετροποίησης. Ο χρήστης έχει τη δυνατότητα να δώσει προκαταβολικά κάποιες ρυθμίσεις όπως το path του backend, τα ονόματα των αλγορίθμων, ορισμένες ετικέτες και στήλες για την εισαγωγή των δεδομένων. Επιπλέον μπορεί να ρυθμίσει την εμφάνιση του σχεδιασμού του γραφήματος

V. Τελικό αποτέλεσμα



VI. Μελλοντικά σχέδια

Μια σειρά μελλοντικών βελτιώσεων έχει ήδη προγραμματιστεί για να ωθήσει το λογισμικό στο επόμενο επίπεδο. Μερικά από αυτά είναι:

- Χρήση αλγορίθμων μηχανικής μάθησης με σκοπό την εκπαίδευση μοντέλου από πολλά ορθώς κατανεμημένα γραφήματα ώστε να γίνεται αυτοματοποιημένη δημιουργία μοντέλων που θα οργανώνουν τη διάταξη σε νέα, χαοτικά γραφήματα σε μια επιφάνεια
- Δυνατότητα μετακίνησης στοιχείων γραφικών με ποντίκι, αφαίρεση ή προσθήκη νέων
- Υποστήριξη χρονικά μεταβαλλόμενων γράφων. Προβολή μιας χρονικής γραμμής δημιουργίας κόμβου και άκρων σε μια κινούμενη προεπισκόπηση για την εξέλιξη των συνδέσεων των κόμβων σε κάποιο γράφημα που παρέχει timestamps στα αρχεία εισαγωγής.
- Ανάπτυξη λογισμικού σε διαδικτυακή εφαρμογή με τη βοήθεια web assembly

Ο κώδικας που δημιουργήθηκε για αυτή την εργασία είναι διαθέσιμος στο github με άδεια GPL και είναι χωρισμένος στο Frontend https://github.com/moreorem/graphdener και backend https://github.com/moreorem/graphdener-backend κομμάτι.

VII. Συμπεράσματα

Η χρήση των δύο συγκεκριμένων γλωσσών προγραμματισμού αποδείχθηκε εξαιρετικά αποτελεσματική για τη δημιουργία ενός συστήματος προβολής και διαχείρισης γραφημάτων. Για να είναι καλύτερο το τελικό αποτέλεσμα απαιτείται ο σωστός διαμοιρασμός καθηκόντων στην κάθε γλώσσα αλλά και η χρήση των κατάλληλων εργαλείων/βιβλιοθηκών. Στη συγκεκριμένη περίπτωση η Python3 σε συνδυασμό με τη βιβλιοθήκη VisPy αποτέλεσε έναν αρκετά κατανοητό τρόπο να σχεδιαστούν οι γραφικές αναπαραστάσεις από αρκετά χαμηλό επίπεδο προγραμματισμού με ταχύτατη ανταπόκριση λόγω της υποστήριξης OpenGL και παράλληλης επεξεργασίας της κάρτας γραφικών.

Η χρήση της Rust επίσης αποδείχθηκε αποτελεσματική καθώς βρίσκεται πλέον σε ένα αρκετά ώριμο επίπεδο με πολλά πλήρη API και βιβλιοθήκες όπως η ArrayFire και το Thread Programming για παραλληλότητα και παραμετροποίηση, καθώς και διάφορα APIs ενδοεπικοινωνίας με άλλες γλώσσες όπως message parsing και remote procedure calls. Η λειτουργία της ειδικά σε release επίπεδο είναι εξαιρετικά γρήγορη ενώ το σύστημα εξαρτήσεων που έχει καθιστά το τελικό έργο τουλάχιστον λειτουργικό και χωρίς προβλήματα.

Η ανταπόκριση του προγράμματος σε τάξεις γραφημάτων έως 1000 αιχμών ήταν άμεση και κατά την εισαγωγή των δεδομένων αλλά και κατά την εφαρμογή των αλγορίθμων διάταξης. Η απόδοση μειώνεται αισθητά όταν το γράφημα αγγίξει τις 100.000 έως 300.000 αιχμές ενώ πλέον χρονοβόρα η διαδικασία γίνεται όταν οι αιχμές ξεπερνούν το εκατομμύριο. Σημειωτέον το γεγονός ότι ο βαθμός ενός γραφήματος έχει αντίκτυπο μόνο στην απόδοση της εφαρμογής του δυναμικά κατευθυνόμενου αλγορίθμου.

Το συγκεκριμένο πρόγραμμα είναι πιο αποδοτικό στην εφαρμογή των αλγορίθμων και στην εισαγωγή των δεδομένων από αρχεία καθώς και στην περιήγηση της προβολής του τελικού γραφήματος συγκριτικά με την ευρέως γνωστή αντίστοιχη βιβλιοθήκη NetworkX [11]. Η διαφορά είναι ακόμα μεγαλύτερη όταν πρόκειται για γραφήματα άνω των 1000 κόμβων.

VIII. Βιβλιογραφία

[1] C. Alsina, *Οι χάρτες του μετρό και τα νευρωνικά δίκτυα (Η θεωρία των γραφημάτων)*. τέσσερα πι, 2012.
[2] *VisuAlgo visualising data structures and algorithms through animation*. .
[3] S. Shanker, “Safe Concurrency with Rust,” 04-Jun-2018. .
[4] “PyQt5 Reference Guide.” [Online]. Available: http://pyqt.sourceforge.net/Docs/PyQt5/.
[5] A. Munshi, D. Ginsburg, and D. Shreiner, *The OpenGL ES 2.0 programming guide*. Upper Saddle River, NJ: Addison-Wesley, 2009.
[6] “Rust Documentation - Defining an enum.” .
[7] “IndraDB - A graph database written in rust.” [Online]. Available: https://github.com/indradb/indradb.
[8] C. Mueller, D. Gregor, and A. Lumsdaine, “Distributed Force-Directed Graph Layout and Visualization,” p. 8.
[9] M. J. McGuffin, “Simple Algorithms for Network Visualization: A Tutorial,” *Tsinghua Sci. Technol.*, p. 16, 2012.
[10] M. E. Belviranlı, “A CIRCULAR LAYOUT ALGORITHM FOR CLUSTERED GRAPHS.”
[11] “NetworkX Software for complex networks.” [Online]. Available: https://networkx.github.io/.
[12] B. Griffen, “Griff’s Graphs.” .
[13] “Concepts: Relational to Graph.” [Online]. Available: https://neo4j.com/developer/graph-db-vs-rdbms/.
[14] “Network Repository.” [Online]. Available: http://networkrepository.com/networks.php.
[15] “Stanford Large Network Dataset Collection.” [Online]. Available: https://snap.stanford.edu/data/.
[16] M. Vedanayaki, “Graph Mining Techniques, Tools and Issues - A Study,” p. 3.
[17] “Vedanayaki - Graph Mining Techniques, Tools and Issues - A Stud.pdf.” .
[18] “McGuffin - 2012 - Simple Algorithms for Network Visualization A Tut.pdf.” .
[19] “Mining approximate frequent patterns from Graph Databases.” .
[20] H. Yifan, “Efficient and high quality force-directed drawing.pdf.” Wolfram Research Inc.
[21] H. Hongmei, “Experiments and Optimal Results for Outerplanar Drawings of Graphs .pdf.”
[22] “ArrayFire.” [Online]. Available: http://arrayfire.org.
[23] B. L. Chamberlain, “Graph Partitioning Algorithms for Distributing Workloads of Parallel Computations,” p. 32.