National and Kapodistrian University of Athens

Department of Informatics and Telecommunications

Point-set Polygonization (CGAL-C++)

Authors:
PSYCHARIS EVANGELOS 1115201800217
SKORDAS CHARISIS 1115201900342

Supervisor:
DR.EMIRIS

An Assignment submitted for the NKUoA:

(DI352) Software Development for Algorithmic Problems

Fall Semester, 2022-23

1 Εισαγωγή

1.1 Πολυγωνοποίηση

Αρχικά κάναμε prototyping σε Python. Υλοποιήσαμε και τους 3 αλγορίθμους (incremental, convex_hull, onion) και τον pick σε μια πρώτη όχι ιδιαίτερα bug-free εκδοχή, σε python, καθώς υπήρχε προηγούμενη εμπειρία και κώδικας από το μάθημα Υπολογιστική Πολυπλοκότητα.

Στη συνέχεια υλοποιήσαμε σε C++/CGAL τους αλγορίθμους incremental, convex hull και onion. Ο onion υλοποιήθηκε μεταγενέστερα, κατά την διάρκεια της δεύτερης εργασίας. Υλοποιήσαμε και τον Pick σε C++/CGAL με έναν απλοϊκό τρόπο αλλά επιλέξαμε να μην τον χρησιμοποιήσουμε (δείτε και το 2.7 pick).

1.2 Βελτιστοποίηση

Από τους αλγορίθμους βελτιστοποίησης υλοποιήσαμε τον local search και τον simulated annealing.

1.3 Επιπλέον κώδικας σε Python και Bash

Visualization App (Python):

Υλοποιήσαμε ένα πρόγραμμα σε python το οποίο από διάφορα αρχεία, που παράγουν κατά την εκτέλεση τους οι αλγόριθμοι, δημιουργούν οπτικοποιήσεις (περισσότερα στο 5.3 vis_app.py)

Σας στέλνουμε μαζί και ένα μικρό video με μια ενδεικτική χρήση του προγράμματος οπτικοποίησης.

Στις υλοποιήσεις των αλγορίθμων, σε διάφορα σημεία, υπάρχει κώδικας σχετικός με οπτικοποίηση. Αυτός συνοδεύεται από σχόλια και if statements με την λέξη ή μεταβλητή vis. Ο κώδικας αυτός δεν εκτελείται εκτός και αν στην κλήση του ./to_polygon προστεθεί η παράμετρος -vis <1/2>, 1 για πλήρη οπτικοποίηση (κάθε βήμα), 2 για σημεία και τελική πολυγωνική γραμμή.

Run Multiple scripts (Python):

Υλοποιήσαμε σε Python 2 scripts για την εκτέλεση των αλγορίθμων σε πολλαπλά αρχεία και με διάφορους συνδυασμούς παραμέτρων, ένα script για πολυγονοποίηση και ένα για βελτιστοποίηση.

Τα scripts αυτά παράγουν .csv αρχεία που χειριζόμαστε ώστε να εξάγουμε πληροφορίες για τους χρόνους εκτέλεσης και τα την ποιότητα των πολυγώνων που παράγουν οι αλγόριθμοι.

Analyse Data (Python/Jupyter):

Δημιουργήσαμε ένα .ipynb στο οποίο κάναμε ανάλυση των δεδομένων από τα προαναφερθέντα .csv. Σε αυτό θα βρείτε γραφικές παραστάσεις και σχόλια για την ποιότητα των αλγορίθμων.

Call CMake (bash):

Το συγκεκριμένο script καλεί το CMake με κατάλληλες παραμέτρους (βιβλιοθήκες CGAL και release build type).

Run (bash):

Η εκτέλεση του προγράμματος άρχισε να γίνεται δαιδαλώδης καθώς επιλέξαμε να κρατήσουμε τις παραμέτρους πολυγωνοποίησης και να προσθέσουμε αυτές τις βελτιστοποίησης ώστε να κάνουμε διάφορες δοκιμές. Μια εκτέλεση του αλγορίθμου (πολυγωνοποίησης και βελτιστοποίησης μαζί) μπορεί να έχει έως 10 παραμέτρους. Η εντολή που προκύπτει είναι μακροσκελής και δυσανάγνωστη.

Αποφασίσαμε λοιπόν να φτιάξουμε ένα script σε bash στο οποίο μπορεί κανείς να ανοίξει το .sh και να διαλέξει τις παραμέτρους για την εκτέλεση. Στη συνέχεια δημιουργείται αυτόματα η εντολή εκτέλεσης. Η εκτέλεση γίνεται καλώντας την εντολή ./run.sh.

1.4 GitHub

Τον κώδικα που σας στέλνουμε μπορείτε επίσης να τον βρείτε στο github στο repository:
https://github.com/Poympas/project-2022
Το παραδοτέο είναι ο φάκελος final_no_cmake:
https://github.com/Poympas/project-2022/tree/main/final_no_cmake
Οι υπόλοιποι φάκελοι περιέχουν παλαιότερο ή δοκιμαστικό κώδικα.

2 Αρχεία κώδικα και περιγραφή

Στο παραδοτέο θα βρείτε την main.cpp το CMakeLists, και το script call_cmake.sh. Θα βρείτε επίσης τον φάκελο includes που περιέχει υποφακέλους, έναν για κάθε module που δημιουργήσαμε. Κάθε υποφάκελος περιέχει μια κεφαλίδα (.hpp), ένα αρχείο πηγαίου κώδικα (.cpp) και το αντίστοιχο CMakeLists. Παρακάτω παραθέτουμε μια σύντομη περιγραφή για κάθε module και τις συναρτήσεις που περιέχει. Πιο αναλυτική περιγραφή μπορείτε να βρείτε στον σχολιασμό του κώδικα σε κάθε αρχείο.

2.1 Γενικά

2.1.1 cgal_config

Στο header γίνεται η επιλογή και το include του kernel της CGAL και περιέχονται typedefs για διάφορους τύπους της CGAL που χρησιμοποιήσαμε (π.χ. Point_2, Segment_2, …). Επιλέξαμε τον kernel Exact_predicates_inexact_constructions_kernel διότι επαρκεί για τους αλγορίθμους που κατασκευάσαμε. Γίνεται επίσης typedef του long long int ως NUM. Ο τύπος NUM χρησιμοποιείται κυρίως για την αποθήκευση εμβαδών. Αρχικά δοκιμάσαμε int αλλά προέκυπταν μεγαλύτεροι αριθμοί. Το αρχείο .cpp δεν περιέχει κάποιον κώδικα, απλά χρειάζεται για το cmake.

2.1.2 io_manip

Το module περιέχει συναρτήσεις για τον χειρισμό του input και του output (input output manipulation). Συγκεκριμένα περιέχει τις παρακάτω συναρτήσεις.

io_manip::read_data
Διαβάζει από ένα αρχείο με ένα σύνολο σημείων τα σημεία και το εμβαδόν του κυρτού τους περιβλήματος. Το αρχείο υποθέτουμε ότι έχει την μορφή που έχουν αυτά που μας δόθηκαν στον φάκελο instances. Δηλαδή, πρώτη γραμμή τίτλος, δεύτερη γραμμή εμβαδόν κυρτού περιβλήματος και στις υπόλοιπες γραμμές τα σημεία, ένα ζευγάρι συντεταγμένων σε κάθε γραμμή.

io_manip::process input
Από τα argc και argv ελέγχει αν έχει γίνει σωστή κλήση του to_polygon και εξάγει το αρχείο εισόδου και εξόδου, τον αλγόριθμο πολυγωνοποίησης και βελτιστοποίησης και τις επιλογές που δόθηκαν για τον καθένα.

io_manip::create_output
Με είσοδο διάφορα στοιχεία όπως η πολυγωνική γραμμή, το εμβαδόν του πολυγώνου και τον χρόνο εκτέλεσης, δημιουργεί ένα string για να γραφτεί στο αρχείο εξόδου με την μορφή που ζητείται στην εκφώνηση.

ΠΑΡΑΤΗΡΗΣΗ : Στο output περιλαμβάνουμε επιπλέον το εμβαδόν που υπολογίζει η CGAL για την πολυγωνική γραμμή (cgal_area) για σύγκριση με το εμβαδόν που υπολογίζουμε κατά την εκτέλεση του αλγορίθμου. Οι δύο αριθμοί πρέπει να είναι ίδιοι. Περιλαμβάνουμε και το pick_calculated_area με τιμή πάντα -1 γιατί όπως είπαμε δεν τον χρησιμοποιήσαμε. Επίσης εμφανίζουμε τα πεδία is simple (με συνάρτηση της CGAL), points και points on poly. Όλα αυτά τα επιπλέον πεδία βοηθήσανε στο debugging και είναι χρήσιμες ενδείξεις για την σωστή εκτέλεση του αλγορίθμου. Αντίστοιχες πληροφορίες εμφανίζονται και για την βελτιστοποίηση με την προσθήκη της αλλαγής στον λόγο εμβαδόν πολυγώνου προς εμβαδόν κυρτού περιβλήματος μετά την βελτιστοποίηση.

io_manip::save_points_to_file
Σώζει τα σημεία που δόθηκαν σε ένα αρχείο. Χρησιμοποιείται για οπτικοποίηση.

io_manip::save_points_and_vis_to_file
Σώζει τα σημεία και μια τιμή bool για κάθε σημείο στο αρχείο που δόθηκε. Χρησιμοποιείται για οπτικοποίηση.

io_manip::print_points
Εκτυπώνει τα δοσμένα σημεία. Χρησιμοποιήθηκε κυρίως για debugging.

2.1.3 visibility

Το module περιέχει συναρτήσεις σχετικές με την ορατότητα, όπως για τον έλεγχο ορατότητας ακμής από σημείο δεδομένης μιας πολυγωνικής γραμμής, και γενικότερα υπορουτίνες που χρησιμοποιούνται από πολλούς αλγόριθμους.

visibility::find_inner_points
Από δύο σύνολα σημείων βρίσκει ποια ανήκουν στο πρώτο και όχι στο δεύτερο. Την χρησιμοποιούμε με πρώτο σύνολο όλα τα σημεία και δεύτερο το κυρτό περίβλημα, οπότε ουσιαστικά υπολογίζονται τα εσωτερικά σημεια.

visibility::choose_index
Η συνάρτηση παίρνει ως είσοδο μια λίστα με εμβαδά τριγώνων και μια λίστα με bools ορατότητας που αντιστοιχούν σε ζευγάρια σημείων και ακμών. Παίρνει ακόμα έναν τρόπο επιλογής ακμής (τυχαία, ελάχιστο/μέγιστο εμβαδόν). Στη συνέχεια διαλέγει και επιστρέφει την θέση στη λίστα ενός ζευγαριού που η ακμή είναι ορατή από το σημείο με βάση την επιλεγμένη στρατηγική.

visibility::are_intersecting
Ελέγχει αν δύο ακμές τέμνονται.

visibility::is_visible_p_from_e
Ελέγχει αν μια ακμή είναι ορατή από ένα σημείο δεδομένης μιας πολυγωνικής γραμμής που μπορεί να εμποδίζει την ορατότητα. Ο έλεγχος γίνεται ελέγχοντας αν τρία ευθύγραμμα τμήματα τέμνονται με οποιαδήποτε από τις ακμές της πολυγωνικής γραμμής. Αν δεν υπάρχει καμία τομή, η ακμή είναι ορατή. Τα τρία ευθύγραμμα τμήματα είναι (σημείο, αρχή ακμής), (σημείο,μέσο ακμής), (σημείο, τέλος ακμής).

visibility::can_connect_polys_simple
Ελέγχει αν δύο πολύγωνα μπορούν να ενωθούν και επιστρέφει πιθανές γέφυρες. Χειρίζεται την περίπτωση που ένα ή και τα δύο πολύγωνα έχουν ένα μόνο σημείο. Και καλείται στις ακραίες αυτές περιπτώσεις από την can_connect_polys.

visibility::can_connect_polys
Ελέγχει αν δύο πολύγωνα μπορούν να ενωθούν και επιστρέφει πιθανές γέφυρες. Μια γέφυρα δύο πολυγώνων είναι ένα ζευγάρι ακμών που ξεκινούν από το μια ακμή ενός πολύγωνο και καταλήγουν σε μια ακμή του άλλου. Οι δύο ακμές της γέφυρας δεν τέμνονται με καμία ακμή των δύο πολυγώνων. Επίσης υποθέτουμε ότι τα πολύγωνα είναι απλά και δεν τέμνονται.

visibility::get_from_to
Επιστρέφει ένα reindexed πολύγωνο που ξεκινάει από ένα δοσμένο σημείο του και καταλήγει σε ένα άλλο, επίσης δοσμένο, σημείο του. Χρησιμοποιείται ως υπορουτίνα στην connect_polys_at_bridge.

visibility::connect_polys_at_bridge
Συνδέει δύο πολύγωνα σε μια δοσμένη γέφυρα.

2.1.4 main

Περιέχει την βασική εκτέλεση του προγράμματος:

1. Διαβάζει, ελέγχει και εξάγει το input από την κλήση ./to_polygon.
2. Διαβάζει τα σημεία και το εμβαδόν του κυρτού πολυγώνου από το αρχείο εισόδου.
3. Καλεί και χρονομετρά τον αλγόριθμο πολυγωνοποίησης με τις δοσμένες παραμέτρους.
4. Καλεί και χρονομετρά τον αλγόριθμο βελτιστοποίησης με τις δοσμένες παραμέτρους.
5. Γράφει στο αρχείο εξόδου τα αποτελέσματα της εκτέλεση.

2.2 Πολυγωνοποίηση

2.2.1 poly_from_ch

Το module περιέχει μια υλοποίηση του αλγορίθμου convex_hull για την εύρεση μιας πολυγωνικής γραμμής.

poly_from_ch::point_closest_to_edge
Από μια ακμή και ένα σύνολο σημείων βρίσκει την κοντινότερη ακμή στο σημείο.

poly_from_ch::remove_point
Διαγράφει ένα σημείο από ένα σύνολο σημείων.

poly_from_ch::run
Εκτελεί τον αλγόριθμο convex hull στα δοσμένα σημεία με την δοσμένη στρατηγική επιλογής ακμής. Πρώτα βρίσκουμε το κυρτό περίβλημα, αρχικοποιούμε με αυτό την πολυγωνική γραμμή και διακρίνουμε τα εσωτερικά σημεία. Σε κάθε βήμα βρίσκουμε τα κοντινότερα σε κάθε ακμή σημεία και επιλέγουμε να προσθέσουμε ένα με βάση το εμβαδόν του τριγώνου που δημιουργεί με την αντίστοιχη ακμή ή τυχαία (ανάλογα με την στρατηγική επιλογής ακμής).

ΠΑΡΑΤΗΡΗΣΗ : Σε κάθε βήμα δεν ξανα υπολογίζουμε όλα τα κοντινότερα σημεία. Αυτά είναι αποθηκευμένα σε μια λίστα και κάθε φορά υπολογίζουμε τα κοντινότερα σημεία μόνο για τις ακμές που είχαν ως κοντινότερο το σημείο που μπήκε στην πολυγωνική γραμμή στο προηγούμενο βήμα. Αρχικά κάναμε τον υπολογισμό για όλα τα σημεία αλλά με αυτή την αλλαγή ο αλγόριθμος έγινε πάρα πολύ πιο γρήγορος.

2.2.2 poly_incremental

Το module περιέχει μια υλοποίηση του αλγορίθμου incremental για την εύρεση μιας πολυγωνικής γραμμής.

poly_incremental::bb_inc_step
Εκτελεί ένα βήμα του αυξητικού αλγορίθμου. Από το τελευταίο κυρτό περίβλημα, πολυγωνική γραμμή και σημείο που μπήκε σε αυτή, υπολογίζει τα επόμενα και το νέο εμβαδόν της πολυγωνικής γραμμής. Το βήμα γίνεται διαλέγοντας μια κόκκινη ακμή. Δηλαδή μια ακμή πάνω στην πολυγωνική γραμμή που βρίσκεται πίσω από τις ακμές του προηγούμενου κυρτού περιβλήματος, οι οποιες είναι ορατές από το νέο σημείο.

ΠΑΡΑΤΗΡΗΣΗ : Τροποποιήσαμε τον αλγόριθμο ώστε όταν γίνεται επιλογή μιας κόκκινης ακμής, κατά τον έλεγχο της ορατότητας της από το καινούργιο σημείο, εξετάζουμε μόνο τις κόκκινες ακμές για να δούμε αν αυτές εμποδίζουν την ορατότητα της κόκκινης ακμής από το καινούργιο σημείο και όχι όλες τις ακμές του πολυγώνου. Αυτό είχε ως αποτέλεσμα ο αλγόριθμος να γίνει σημαντικά πιο γρήγορος.

poly_incremental::run
Εκτελεί τον αλγόριθμο incremental στα δοσμένα σημεία με την δοσμένη στρατηγική επιλογής ακμής και αρχικοποίησης. Ξεκινάει με το πιο ακραίο τρίγωνο βάση της αρχικοποίησης και προσθέτει σημεία καλώντας την bb_inc_step μέχρι να εξαντληθούν τα σημεία.

2.2.3 poly_onion

Το module περιέχει μια υλοποίηση του αλγορίθμου onion για την εύρεση μιας πολυγωνικής γραμμής.

poly_onion::run
Εκτελεί τον αλγόριθμο onion στα δοσμένα σημεία με την δοσμένη στρατηγική αρχικοποίησης. Βρίσκουμε το κυρτό περίβλημα και τα εσωτερικά σημεία. Βρίσκουμε το κυρτό περίβλημα των εσωτερικών σημείων και τα νέα εσωτερικά σημεία. Συνεχίζουμε έτσι μέχρι να μην έχουμε άλλα σημεία. Συνδέουμε τα κυρτά περιβλήματα από έξω προς τα μέσα.

2.2.4 pick

Το module περιέχει μια υλοποίηση του αλγορίθμου pick για την εύρεση εμβαδού πολυγώνου.

pick::run
Καλεί τον αλγόριθμο pick. Κάναμε από περιέργεια μια απλή υλοποίηση του αλγορίθμου, αν και είμασταν ομάδα δύο ατόμων και δεν ήταν ζητούμενο. Εν τέλει δεν τον χρησιμοποιήσαμε γιατί στην απλή παρούσα μορφή του - έλεγξε για όλα τα σημεία στο bounding box του πολυγώνου αν είναι εντός του πολυγώνου ή πάνω σε αυτό - είναι πολύ αργός.

2.2.5 poly_line_algorithms

Το header περιέχει συναρτήσεις για την κλήση των αλγορίθμων πολυγωνοποίησης και του αλγορίθμου pick. Το αρχείο .cpp δεν περιέχει κάποιον κώδικα, απλά χρειάζεται για το cmake.

poly_line_algorithms::incremental
Καλεί τον αλγόριθμο incremental.

poly_line_algorithms::convex_hull
Καλεί τον αλγόριθμο convex_hull.

poly_line_algorithms::onion
Καλεί τον αλγόριθμο onion.

poly_line_algorithms::pick
Καλεί τον αλγόριθμο pick.

2.3 Βελτιστοποίηση

2.3.1 opt_local_search

Το module περιέχει μια υλοποίηση του local search για την βελτιστοποίηση μιας πολυγωνικής γραμμής.

opt_local_search::run
Εκτελεί τον αλγόριθμο local search στο δοσμένα πολύγωνο με παραμέτρους L, min.max και threshold. Διαλέγουμε L διαδοχικά σημεία που προκαλούν την μέγιστη ή ελάχιστη αλλαγή στο εμβαδόν του πολυγώνου, τα αφαιρούμε από το πολύγωνο και τα τοποθετούμε σε ένα άλλο σημείο του πολυγώνου. Επαναλαμβάνουμε μέχρι η αλλαγή στο λόγο του εμβαδού το πολυγώνου προς το εμβαδόν του κυρτού περιβλήματος να είναι μικρότερη από το threshold.

2.3.2 opt_simulated_annealing

Το module περιέχει μια υλοποίηση του simulated annealing για την βελτιστοποίηση μιας πολυγωνικής γραμμής.

opt_simulated_annealing::local
Εκτελεί τον αλγόριθμο simulated annealing με local step στο δοσμένα πολύγωνο. Κάνουμε L στο πλήθος local steps και τα εφαρμόζουμε ή όχι στο πολύγωνο με βάση το κριτήριο Mitropolis.

opt_simulated_annealing::global
Εκτελεί τον αλγόριθμο simulated annealing με global step στο δοσμένα πολύγωνο. Κάνουμε L στο πλήθος global steps και τα εφαρμόζουμε ή όχι στο πολύγωνο με βάση το κριτήριο Mitropolis.

opt_simulated_annealing::subdiv
Εκτελεί τον αλγόριθμο simulated annealing με subdivision. Σπάμε το πολύγωνο σε υποσύνολα και κάθε υποσύνολο το πολυγωνοποιούμε (με βάση τον δοσμένο αλγόριθμο) και το βελτιστοποιούμε (με simulated annealing και global step). Τέλος συνδέουμε τα πολύγωνα.

opt_simulated_annealing::run
Εκτελεί τον αλγόριθμο simulated annealing στο δοσμένα πολύγωνο με παραμέτρους L, min.max και local/global/subdiv.

2.3.3 opt_algorithms

Το header περιέχει συναρτήσεις για την κλήση των αλγορίθμων βελτιστοποίησης. Το αρχείο .cpp δεν περιέχει κάποιον κώδικα, απλά χρειάζεται για το cmake.

opt_algorithms::local_search
Καλεί τον αλγόριθμο local search.

opt_algorithms::simulated_annealing
Καλεί τον αλγόριθμο simulated annealing.

3 Οδηγίες μεταγλώττισης και χρήσης

Οδηγίες μεταγλώττισης:

1. chmod 777 cal_cmake.sh
2. ./call_cmake.sh
3. make

Οδηγίες εκτέλεσης:

Για την εκτέλεση καλέστε την to_polygon με παραμέτρους πολυγωνοποίησης και βελτιστοποίησης.

Για την πολυγωνοποίηση:

Για τον αλγόριθμο convex_hull:
./to_polygon -i <input_file> -o <output_file> -algorithm convex_hull -edge_selection <1/2/3>

Για τον αλγόριθμο incremental:
./to_polygon -i <input_file> -o <output_file> -algorithm incremental -edge_selection <1/2/3> -initialization <1a/1b/2a/2b>

Για τον αλγόριθμο onion:
./to_polygon -i <input_file> -o <output_file> -algorithm onion -onion_initialization <1a/1b/2a/2b>

Για την βελτιστοποίηση:

Προσθέστε στην εντολή πολυγωνοποίησης ένα από τα παρακάτω:

Για τον αλγόριθμο local search:
-opt_algorithm local_search -minmax <1,2> -L <1 to 10> -threshold <threshold>

Για τον αλγόριθμο simulated annealing:
-opt_algorithm simulated_annealing -minmax <1,2> -L <L> -local_global_subdiv <1,2,3>

Η παράμετρος minmax με τιμή 1 αντιστοιχεί στο min, με 2 στο max.
Η παράμετρος local_global_subdiv με τιμή 1 αντιστοιχεί στον local, με 2 στο global, με 3 στον subdiv.

Παραδείγματα εκτέλεσης:

incremental χωρίς βελτιστοποίηση:
./to_polygon -i input_data/euro-night-0000200.instance -o data.out -algorithm incremental -edge_selection 3 -initialization 1a

convex_hull και local_search:
./to_polygon -i input_data/euro-night-0000050.instance -o data.out -algorithm convex_hull -edge_selection 2 -opt_algorithm local_search -minmax 2 -L 10 -threshold 0.0001

incremental και simulated_annealing με local step:
./to_polygon -i input_data/euro-night-0000100.instance -o data.out -algorithm incremental -edge_selection 3 -initialization 1a -opt_algorithm simulated_annealing -minmax 2 -L 5000 -local_global_subdiv 1

run.sh:

Σε αυτό το σημείο η συγγραφή της εντολής για την εκτέλεση του to_polygon έγινε αρκετά χρονοβόρα. Για αυτό τον λόγο φτίαξαμε το bash script run.sh για την διευκόλυνση μας και σας. Για να το χρησιμοποιήσετε ανοίξτε, τροποποιήστε τις παραμέτρους σε αυτές που επιθυμείτε να δοκιμάστε και τρέξτε την εντολή ./run.sh. Κατά πάσα πιθανότητα θα χρειαστεί πρώτα να τρέξετε και την εντολή chmod 777 run.sh.

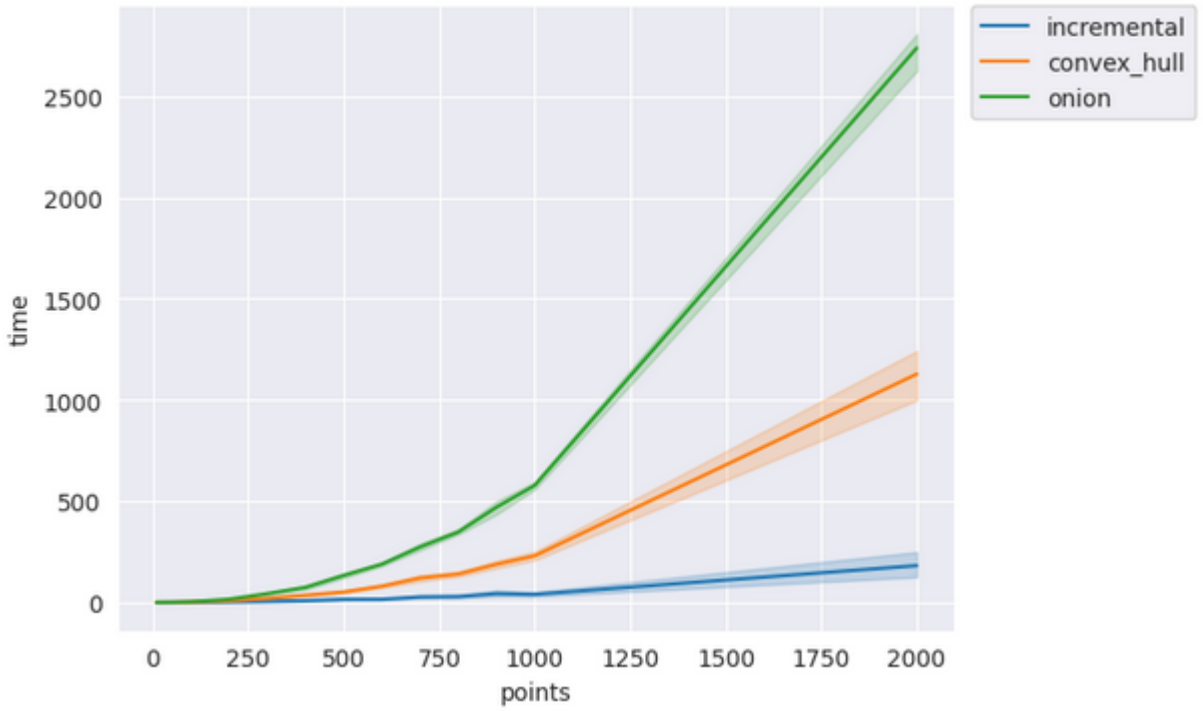
Γενικά σας συνιστούμε είτε να χρησιμοποιήσετε το script είτε να τροποποιήσετε τα προαναφερθέντα παραδείγματα για να εκτελέσετε παραδείγματα της επιλογής σας.

4 Σχολιασμός αποτελεσμάτων

Δημιουργήσαμε ένα .ipynb με στο οποίο φτιάξαμε γραφικές παραστάσεις για την ανάλυση των αποτελεσμάτων από την εκτέλεση των αλγορίθμων. Τα σχόλια που θα βρείτε εδώ μπορείτε να τα δείτε και στο analise_data.ipynb.

4.1 Πολυγωνοποίηση

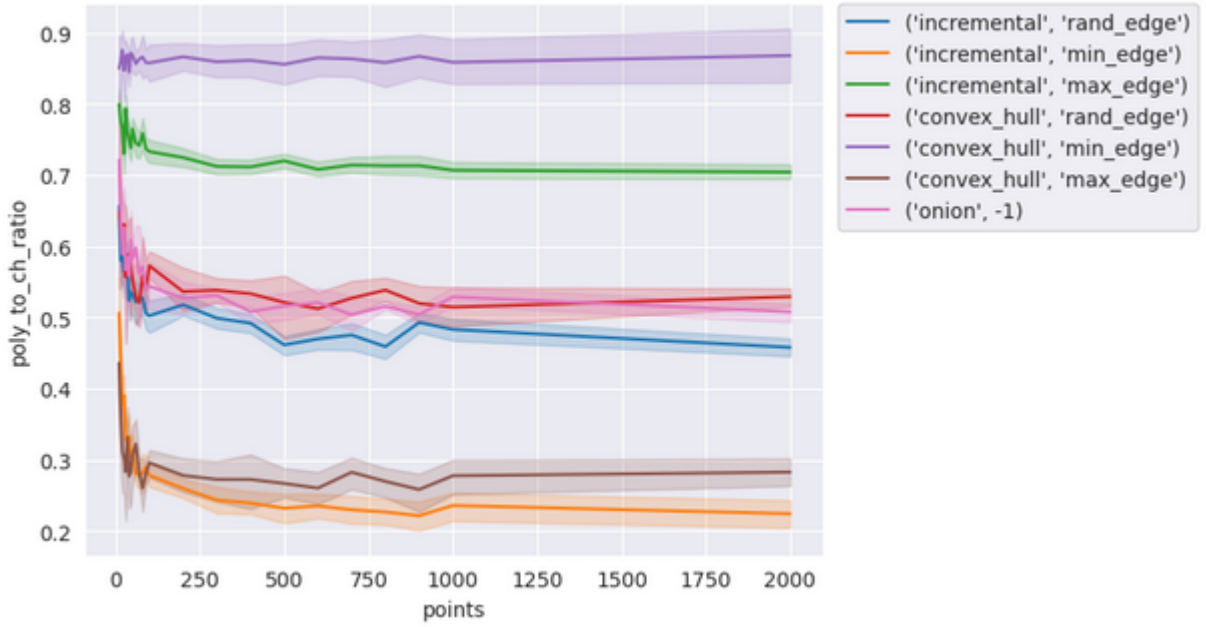
4.1.1 Χρόνοι εκτέλεσης



Παραπάνω βλέπουμε τους χρόνους εκτέλεσης των αλγορίθμων πολυγωνοποίησης για point sets μεγέθους από 10 έως 2000. Στον άξονα y έχουμε τον χρόνο εκτέλεσης σε milliseconds και στον x τον αριθμό των σημείων.

Βλέπουμε ότι ο incremental είναι ο πιο γρήγορος ακολουθούμενος από τον convex hull και πιο αργός είναι ο onion.

4.1.2 Λόγος εμβαδού πολυγώνου προς εμβαδόν κυρτού περιβλήματος



Παραπάνω το ratio των αλγορίθμων πολυγωνοποίησης για point sets μεγέθους από 10 έως 2000. Στον άξονα y έχουμε το ratio και στον x τον αριθμό των σημείων.

Για πολύγωνα μέγιστου εμβαδού παρατηρούμε ότι ο convex hull (μωβ γραμμή - convex_hull,min_edge) φτιάχνει καλύτερα πολύγωνα (περίπου 0.85 ratio) από τον incremental (πράσινη γραμμή - incremental,max_edge) (about 0.7 ratio).

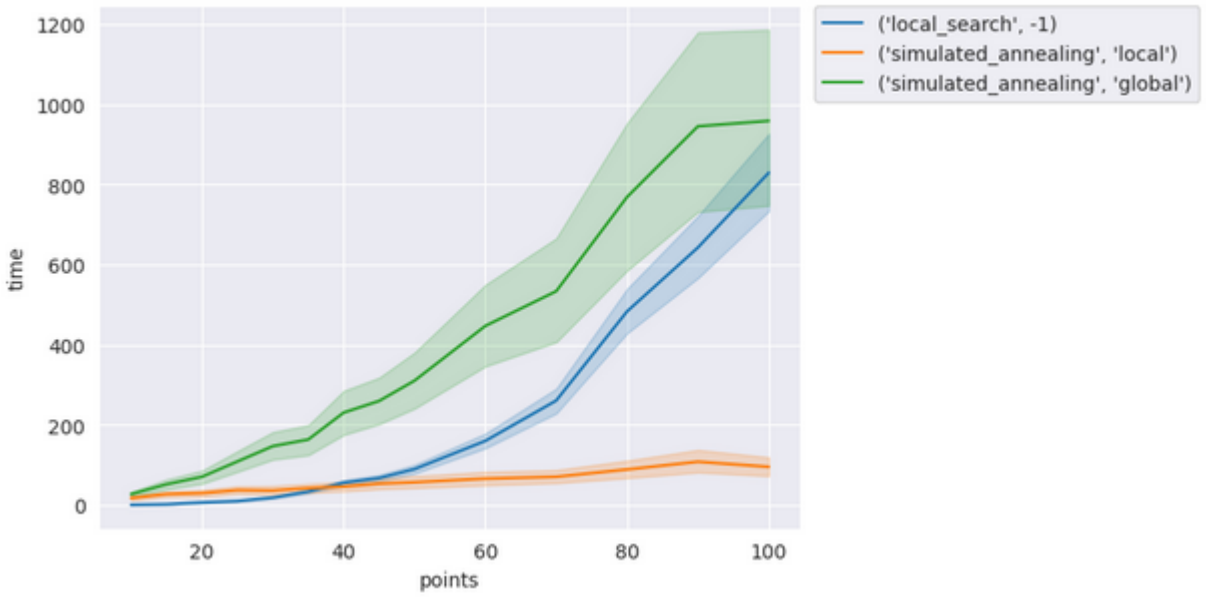
Για πολύγωνα ελάχιστου εμβαδού οι αλγόριθμοι convex_hull (καφέ γραμμή- convex_hull,max_edge) και incremental (πορτοκαλί γραμμή - incremental,min_edge) φτιάχνουν σχεδόν εξίσου καλά πολύγωνα (περίπου 0.7 ratio).
(pink line - onion,-1) makes polygons with ratio of about 0.5. So do the incremental (blue line - incremental,rand_edge)
Οι αλγόριθμοι onion (ροζ γραμμή - onion,-1) και incremental (μπλε γραμμή - incremental,rand_edge) , convex hull (κόκκινη γραμμή - convex_hull,rand_edge) με τυχαία επιλογή ακμης φτιάχνουν όλοι πολύγωνα με ratio περίπου 0.5.

4.2 Βελτιστοποίηση (local search, sim an local/global)

Η ανάλυση του αλγορίθμους simulated annealing με subdivision γίνεται ξεχωριστά καθώς μοιάζει περισσότερο με τους αλγορίθμους πολυγωνοποίησης στην ανάλυση.

Στην παρακάτω ανάλυση χρησιμοποιήθηκε incremental για τα ελάχιστα πολύγωνα και convex_hull για τα μέγιστα.

4.2.1 Χρόνοι εκτέλεσης



Παραπάνω βλέπουμε τους χρόνους εκτέλεσης των αλγορίθμων βελτιστοποίησης για point sets μεγέθους από 10 έως 100. Στον άξονα y έχουμε τον χρόνο εκτέλεσης σε milliseconds και στον x τον αριθμό των σημείων.

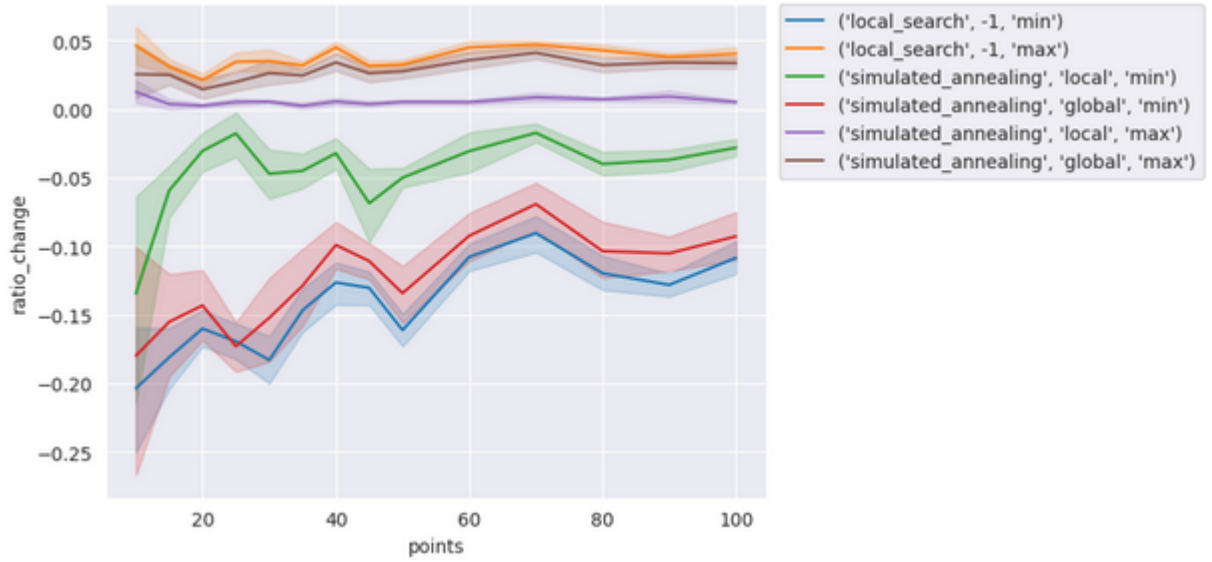
Βλέπουμε ότι ο simulated annealing με local step είναι ο πιο γρήγορος ακολουθούμενος από τον local search και πιο αργός είναι ο simulated annealing με global step.

Γενικά οι αλγόριθμοι είναι πολύ πιο αργοί από τους αλγορίθμους πολυγωνοποίησης. Παρατηρήστε ότι είμαστε ήδη περίπου στο ένα δευτερόλεπτο στα 100 σημεία. Ο μόνος αλγόριθμος που μπορεί ενδεχομένως να χρησιμοποιηθεί σε μεγαλύτερα σημειοσύνολα είναι ο simulated annealing με local search.

Παρατηρήστε επίσης την διακύμανση στον local step εξαιτίας της παραμέτρου L (από 1 έως 10). Δεν είναι ιδιαίτερα μεγάλη διότι καθώς μεγαλώνει το L η πιθανότητα να βρούμε μια αλλαγή που να μην χαλάει την απλότητα του πολυγώνου μειώνεται. Ενδεχομένως να είναι εντάξει να χρησιμοποιούμε L=10 σε κάθε περίπτωση.

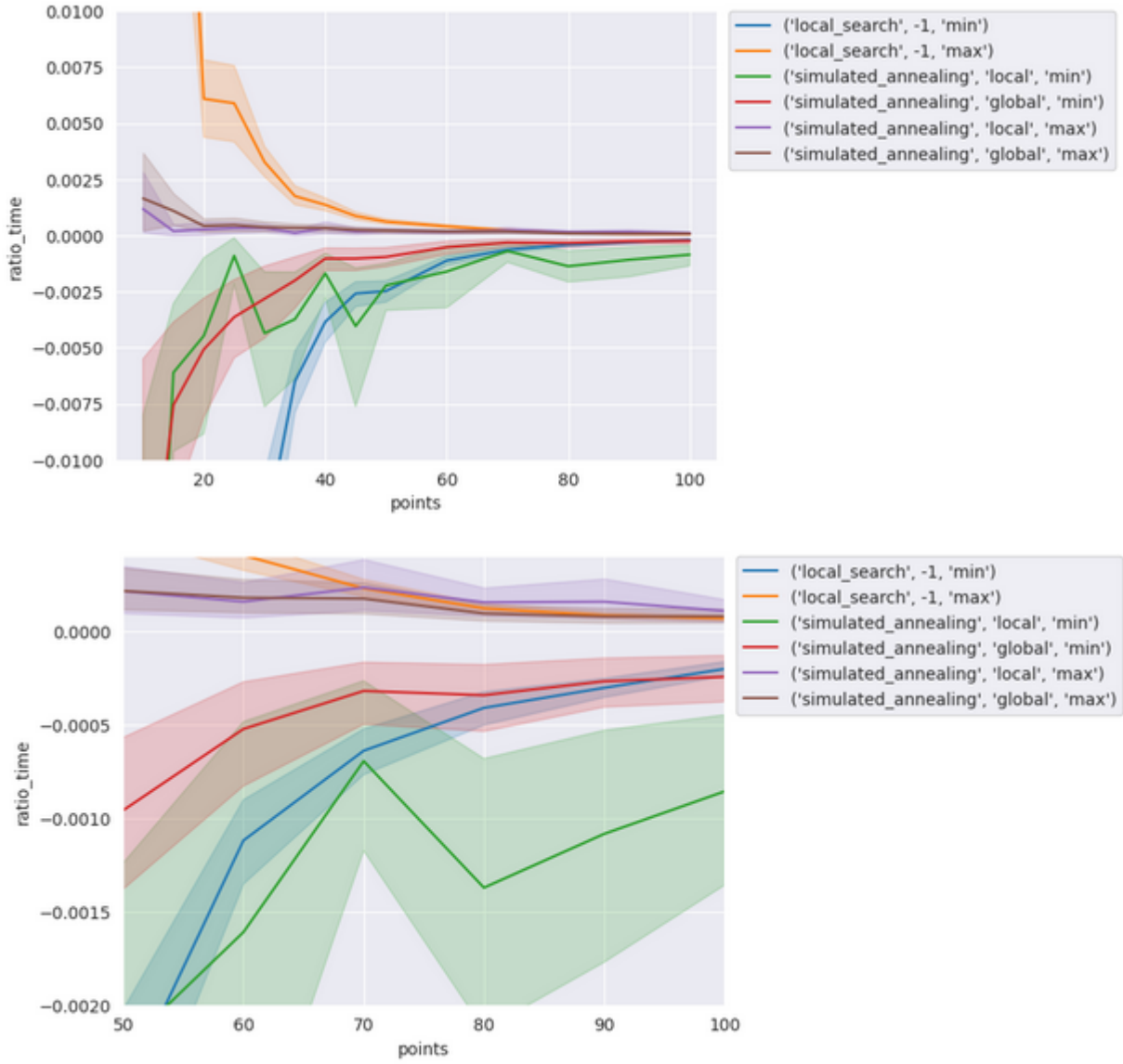
Τέλος η διακύμανση στον simulated annealing με global step είναι μεγάλη γιατί η παράμετρος L παίρνει τιμές από 1000 έως 10000 και αυτή τη φορά το L είναι ο αριθμός των επαναλήψεων.

4.2.2 Αλλαγή στο ratio



Βλέπουμε ότι για μεγιστοποίηση και ελαχιστοποίηση ο local search (μπλε και πορτοκαλί γραμμή) δίνει τα καλύτερα αποτελέσματα ακολουθούμενος από τον simulated annealing με global step(καφέ και κόκκινη γραμμή) και χειρότερος είναι ο simulated annealing με local step (πράσινη και μωβ γραμμή).

4.2.3 Αλλαγή στο ratio διαιρεμένη με χρόνο εκτέλεσης



Η δεύτερη γραφική είναι απλώς zoomαρισμένη.

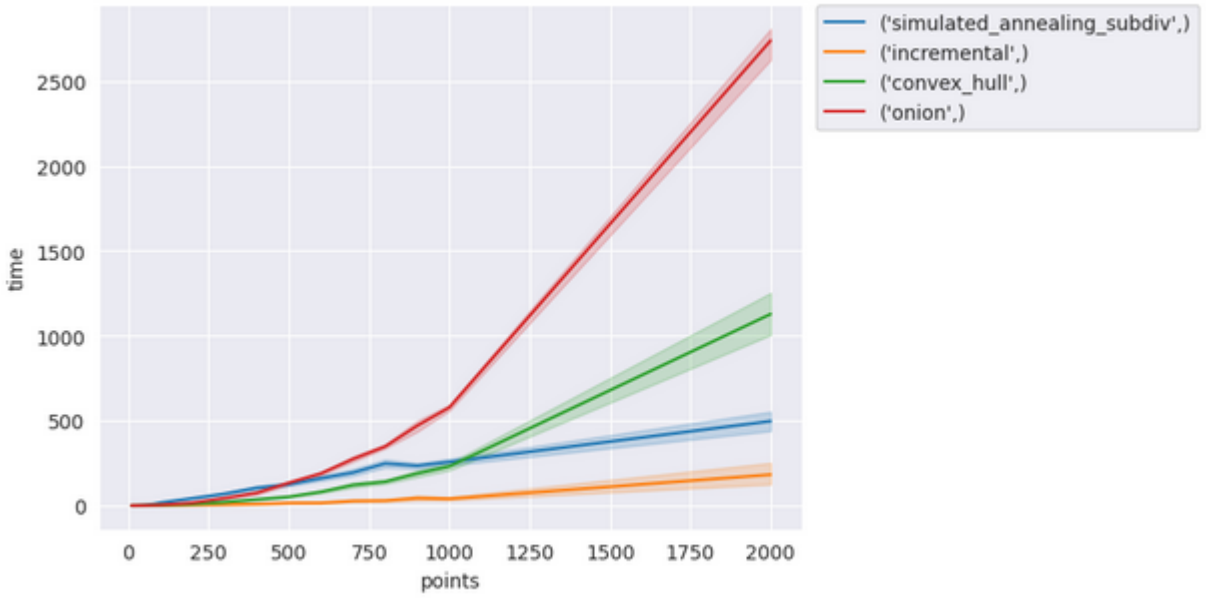
Σκεφτήκαμε ότι μια καλύτερη ένδειξη για την ποιότητα των αλγορίθμων βελτιστοποίησης είναι η αλλαγή στο ratio διαιρεμένη με τον χρόνο εκτέλεσης. Η τιμή αυτή μας δίνει πόσο θα αλλάξει το ratio αν ο αλγόριθμος τρέχει για ένα δευτερόλεπτο.

Τώρα φαίνεται να μην έχει ιδιαίτερη σημασία η επιλογή του αλγορίθμου για βελτιστοποίηση ενώ για ελαχιστοποίηση ο simulated annealing με local step (πράσινη γραμμή) φαίνεται να ξεπερνά τους άλλους δύο. Η μεγάλη του διακύμανση επίσης μαρτυρά ότι έχει νόημα να δώσουμε μεγαλύτερες τιμές στο L, περισσότερο από ότι σε άλλες περιπτώσεις.

4.3 Βελτιστοποίηση (sim an subdiv)

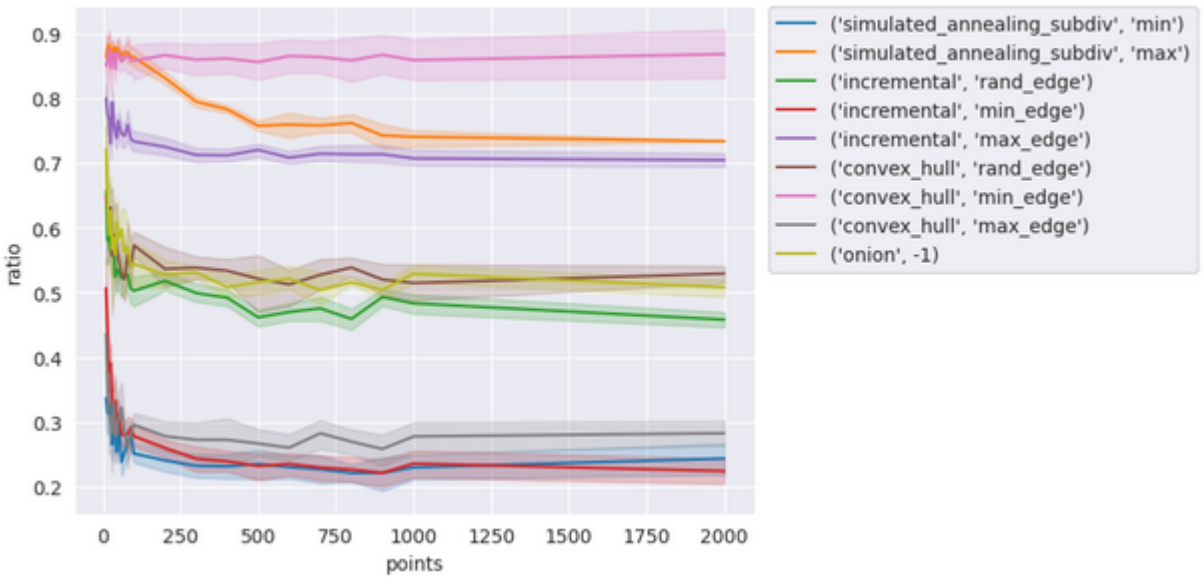
Η ανάλυση του αλγορίθμους simulated annealing με subdivision μοιάζει περισσότερο με τους αλγορίθμους πολυγωνοποίησης στην ανάλυση. Η διαδικασία είναι ότι σπάμε το σημειοσύνολο σε υποσύνολα, πολυγωνοποιούμε κάθε υποσύνολο, βελτιστοποιούμε κάθε υποσύνολο και συνδέουμε τα πολύγωνα που προκύπτουν. Επομένως δεν έχει τόσο νόημα να μελετήσουμε την αλλαγή στο ratio για κάθε υπο πολύγωνο. Επίσης ο αλγόριθμος χρησιμοποιεί simulated annealing με local search, τον οποίο αναλύσαμε προηγουμένως.

4.3.1 Χρόνοι εκτέλεσης



Παραπάνω βλέπουμε ότι ο αλγόριθμος (μπλε γραμμή) είναι λίγο πιο αργός από τον incremental και γρηγορότερος από τον convex_hull και τον onion.

4.3.1 Λόγος εμβαδού πολυγώνου προς εμβαδόν κυρτού περιβλήματος

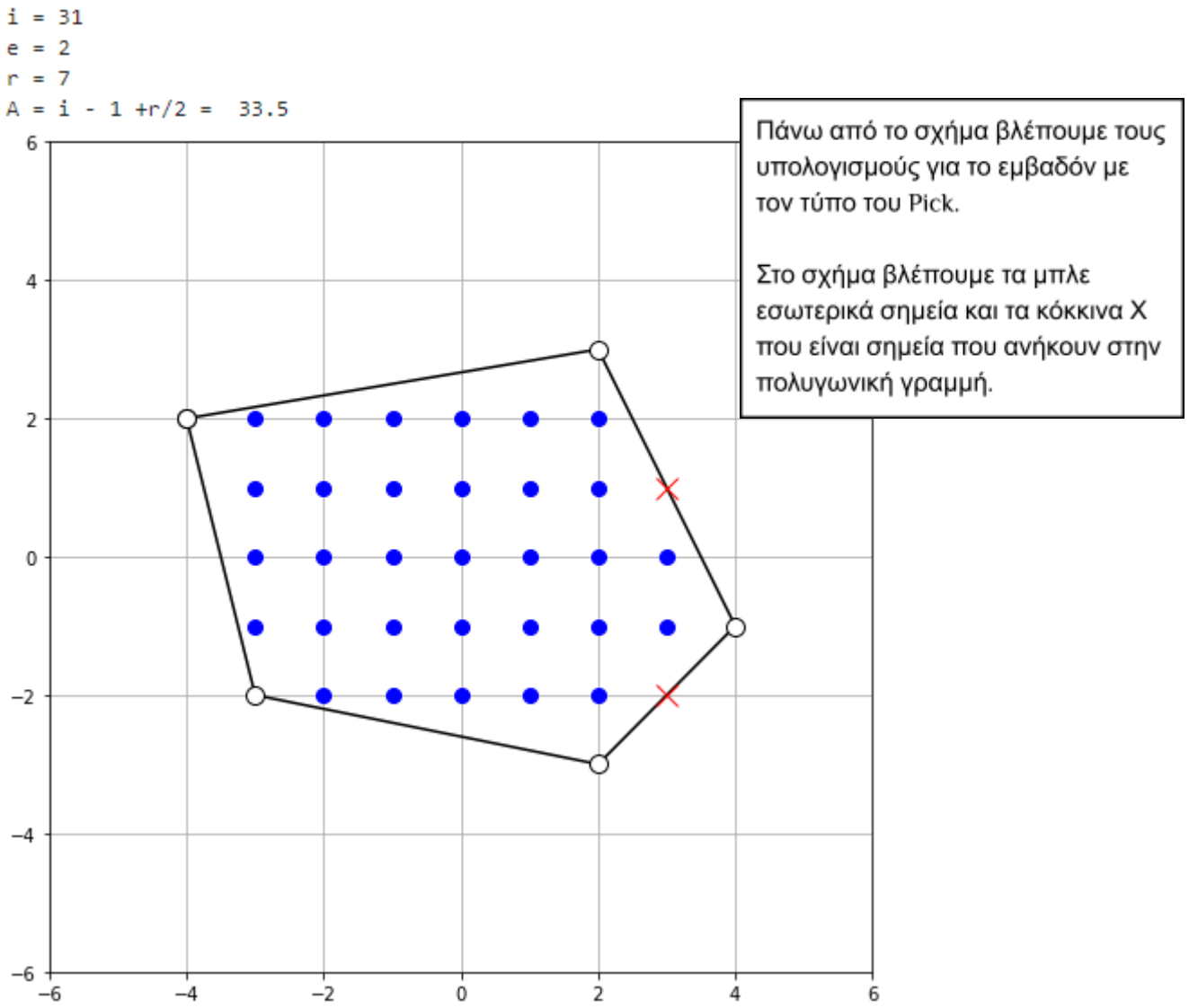


Παραπάνω βλέπουμε ότι ο αλγόριθμος (πορτοκαλί γραμμή) φτιάχνει μεγιστικα πολύγωνα λίγο καλύτερα (περίπου 0.7 ratio) από τον incremental και χειρότερα από τον convex_hull. Επίσης φτιάχνει (μπλε γραμμή) ελαχιστικά πολύγωνα εξίσου καλά με τους incremental και convex hull (περίπου 0.25 ratio).

5 ***Επιπλέον σχόλια και περιεχόμενα***

5.1 pick

Όπως έχουμε ήδη αναφέρει, υλοποιήσαμε και τον pick (σε C++/CGAL και python). Η υλοποίηση του pick υπάρχει σε αντίστοιχο module αλλά όπως αναφέραμε ήδη δεν τον χρησιμοποιήσαμε γιατί η υλοποίηση ήταν αρκετά αργή. Ωστόσο παραθέτουμε μια ενδιαφέρουσα οπτικοποίηση που έγινε σε python.



5.2 vis_app.py

Όπως αναφέραμε στην εισαγωγή, δημιουργήσαμε και ένα πρόγραμμα οπτικοποίησης των αλγορίθμων (πολυγωνοποίσης και βελτιστοποίησης), το οποίο από διάφορα αρχεία που δημιουργούν οι αλγόριθμοι (αν επιλεχθεί η παράμετρος -vis <1/2>) δημιουργεί οπτικοποιήσεις.

Το πρόγραμμα λέγεται vis_app.py και το περιέχουμε και αυτό στο αρχείο που σας στέλνουμε. Στον φάκελο vis_app_includes βρίσκονται διάφορα modules που χρησιμοποιεί.

Χρησιμοποιεί διάφορα πακέτα (όπως matplotlib και PySimpleGUI) που ίσως χρειαστεί να εγκαταστήσετε. Κάποιες εγκαταστάσεις που μπορεί να σας φανούν χρήσιμες:

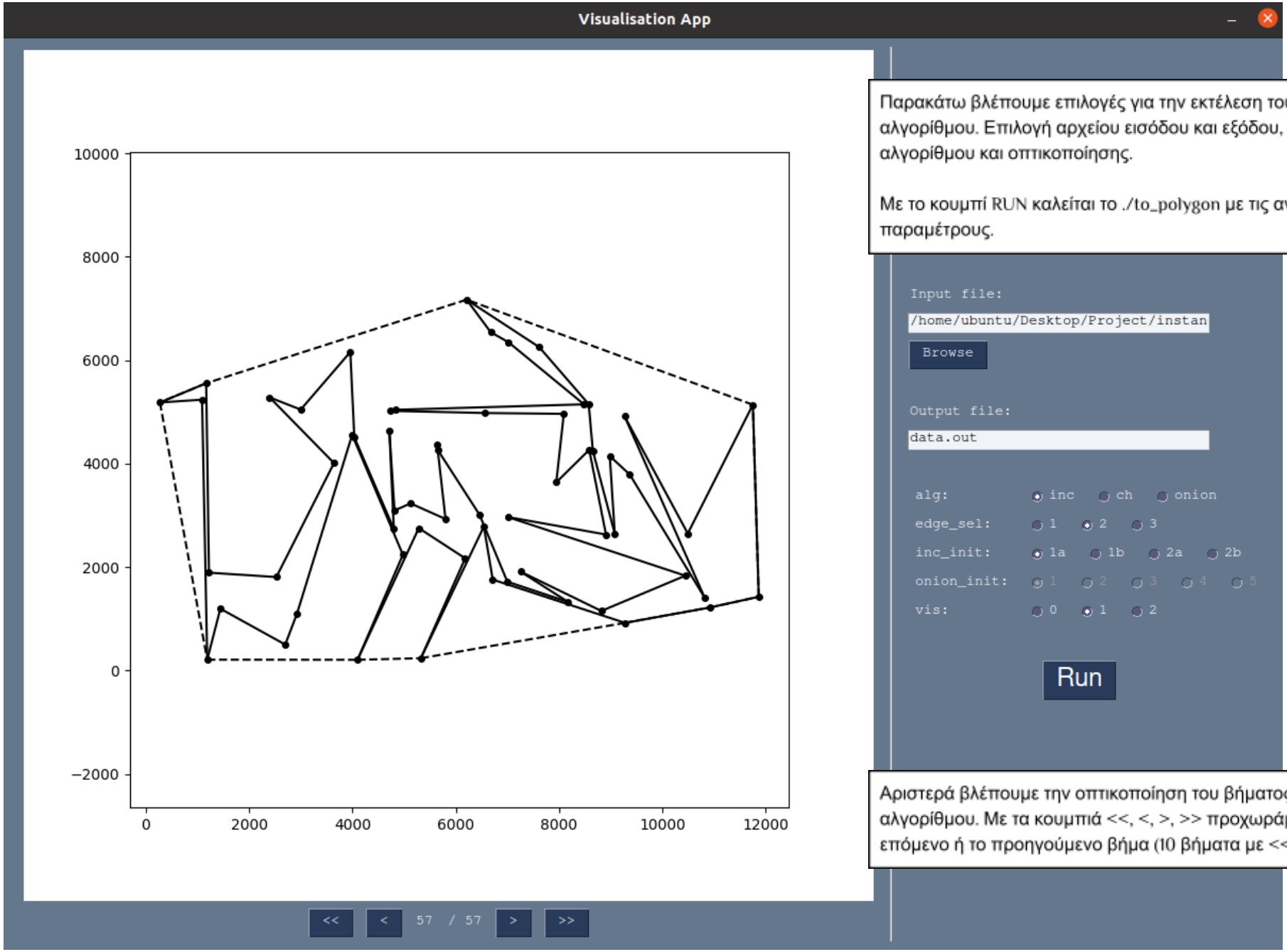
tkinter: sudo apt-get install python3-tk
pil: sudo apt-get install python3-pil
imagetk: sudo apt-get install python3-pil.imagetk

Για την εκτέλεση του προγράμματος αρκεί το vis_app.py να βρίσκεται στον ίδιο φάκελο με το to_polygon και η κλήση γίνεται με την εντολή:

```
python3 vis_app.py
```

Στη συνέχεια παραθέτουμε μερικές ενδεικτικές εικόνες από την εκτέλεση της εφαρμογής vis_app.py.

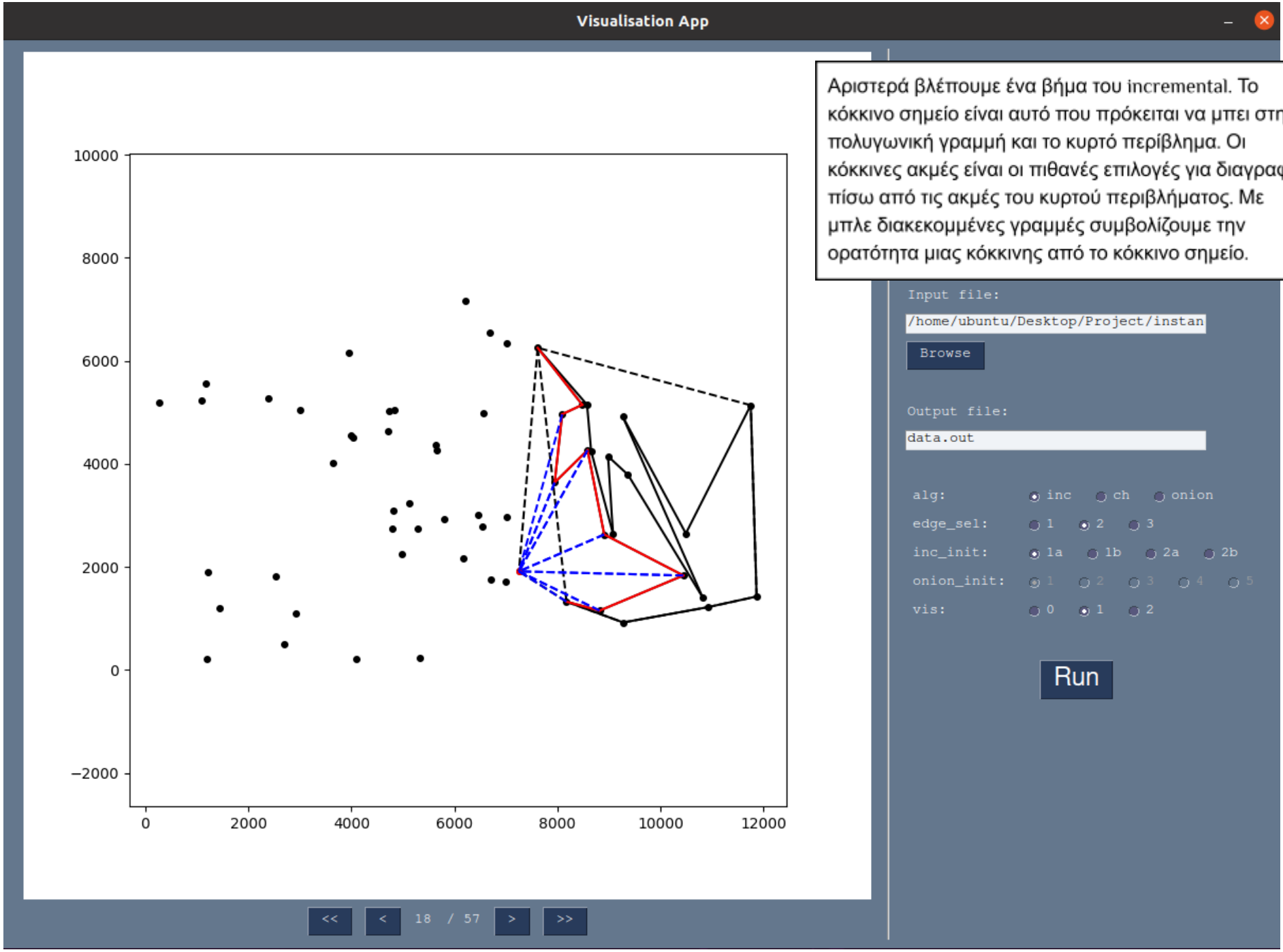
Επίσης όπως αναφέραμε σας στέλνουμε και ένα σχετικό βίντεο με μια ενδεικτική εκτέλεση της εφαρμογής.

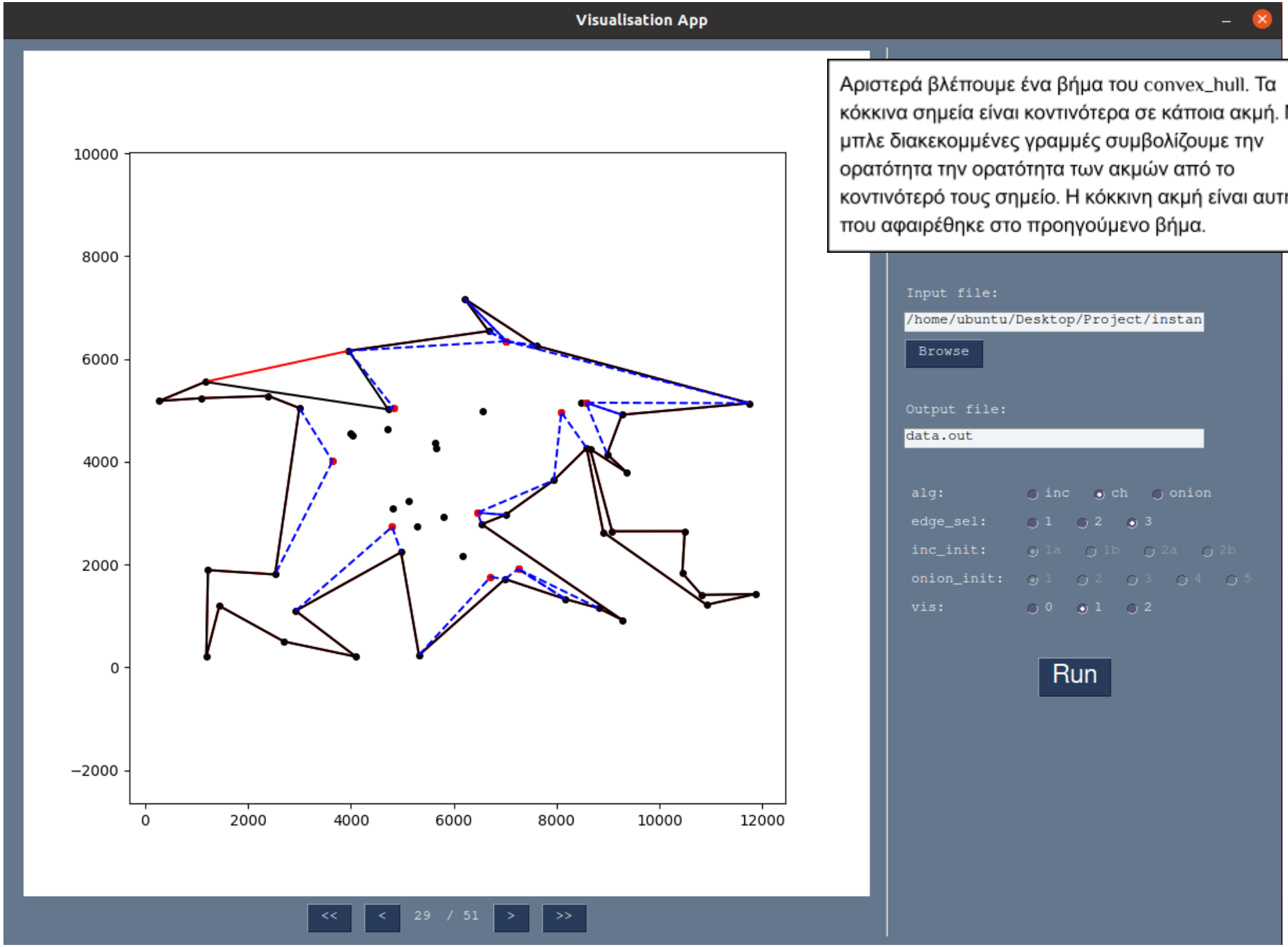


Παρακάτω βλέπουμε επιλογές για την εκτέλεση του αλγορίθμου. Επιλογή αρχείου εισόδου και εξόδου, επιλογές αλγορίθμου και οπτικοποίησης.

Με το κουμπί RUN καλείται το ./to_polygon με τις αντίστοιχες παραμέτρους.

Αριστερά βλέπουμε την οπτικοποίηση του βήματος του αλγορίθμου. Με τα κουμπιά <<, <, >, >> προχωράμε στο επόμενο ή το προηγούμενο βήμα (10 βήματα με << και >>).


Visualisation App
Αριστερά βλέπουμε ένα βήμα του incremental. Το κόκκινο σημείο είναι αυτό που πρόκειται να μπει στην πολυγωνική γραμμή και το κυρτό περίβλημα. Οι κόκκινες ακμές είναι οι πιθανές επιλογές για διαγραφή, πίσω από τις ακμές του κυρτού περιβλήματος. Με μπλε διακεκομμένες γραμμές συμβολίζουμε την ορατότητα μιας κόκκινης από το κόκκινο σημείο.
10000
8000
6000
4000
2000
0
−2000
0
2000
4000
6000
8000
10000
12000
Input file:
/home/ubuntu/Desktop/Project/instan
Browse
Output file:
data.out
alg: inc ch onion
edge_sel: 1 2 3
inc_init: 1a 1b 2a 2b
onion_init: 1 2 3 4 5
vis: 0 1 2
Run
<< < 18 / 57 > >>



Αριστερά βλέπουμε ένα βήμα του convex_hull. Τα κόκκινα σημεία είναι κοντινότερα σε κάποια ακμή. Με μπλε διακεκομμένες γραμμές συμβολίζουμε την ορατότητα την ορατότητα των ακμών από το κοντινότερό τους σημείο. Η κόκκινη ακμή είναι αυτή που αφαιρέθηκε στο προηγούμενο βήμα.

5.3 run_multiple.py & opt_run_multiple.py

Τα δύο αυτά αρχεία εκτελούν αλγορίθμους πολυγονοποίησης και βελτιστοποίησης αντίστοιχα για όλα τα σημειοσύνολα συγκεκριμένων μεγεθών. Τα χρησιμοποιήσαμε για να εξάγουμε δεδομένα από την εκτέλεση των αλγορίθμων και στη συνέχεια να τα αναλύσουμε στο analise_data.ipynb.

Με τα δύο αυτά προγράμματα δημιουργήσαμε τρία data sets τα οποία και συμπεριλαμβάνουμε. Το data.csv, το opt_data.csv και το lgs_data.csv.

5.4 analise_data.ipynb

Με τα datasets data.csv, το opt_data.csv και το lgs_data.csv κάνουμε ανάλυση των αποτελεσμάτων από την εκτέλεση των αλγορίθμων. Για οπτικοποιήσεις χρησιμοποιούμε την βιβλιοθήκη seaborn την οποία μπορεί να χρειαστεί να εγκαταστήσετε αν θέλετε να επανεκτελέσετε το .ipynb, αν και σας το στέλνουμε ήδη εκτελεσμένο.

Για την εγκατάσταση της βιβλιοθήκης seaborn: pip install seaborn

5.5 run.sh

Όπως αναφέρθηκε και στο κεφάλαιο 3 μπορείτε να το χρησιμοποιήσετε για να δειευκολυνθείτε στην εκτέλεση του προγράμματος.