

**Υλοποίηση Αλγορίθμων για την Κάλυψη Σημείων με ένα Ορθογώνιο Μικρού Εμβαδού**

**Σπυρίδων Σιλίρας**

**Διπλωματική Εργασία**

Επιβλέπων Καθηγητής: Λεωνίδας Παληός

Ιωάννινα, Σεπτέμβριος, 2024

**Τμήμα Μηχ. Η/Υ & Πληροφορικής**

**Πανεπιστήμιο Ιωαννίνων**

**Department of Computer Science & Engineering**

**University of Ioannina**

Ευχαριστίες

Θα ήθελα να ευχαριστήσω όλους τους φίλους μου και συμφοιτητές, για τις γόνιμες συζητήσεις που είχαμε καθ’ όλη τη διάρκεια των σπουδών μου κάνοντας πιο χαρούμενη την ολοκλήρωση τους. Επίσης, πολύ σημαντικός ήταν ο ρόλος και το έργο όλων των καθηγητών του τμήματος, που συνέβαλαν έτσι ώστε να κατανοήσω καλύτερα τους διάφορου τομείς της επιστήμης μας, όμως ιδιαίτερες ευχαριστίες οφείλω στον Καθηγητή, κ. Λεωνίδα Παληό, που με καθοδήγησε στην εκπόνηση αυτής της εργασίας με τις σημαντικές παρατηρήσεις και διευκρινήσεις του. Τέλος, θέλω να ευχαριστήσω την οικογένεια μου για την υπομονή και την στήριξη, αλλά και εμπιστοσύνη που μου έδειξαν όλα αυτά τα χρόνια.

Σεπτέμβριος, 2024

Σπυρίδων Σιλίρας

Περίληψη

Θεωρούμε ένα σύνολο P των n σημείων στον χώρο. Δοθέντος ενός ακέραιου k > 0, δείχνουμε πως υπολογίζουμε ένα ορθογώνιο, παράλληλο στους άξονες, με το μικρότερο εμβαδό, που περιέχει k σημεία, σε χρόνο O(n k2 log n + n log2 n). Επίσης, θεωρούμε το πρόβλημα για το οποίο δοθέντος μιας τιμής α > 0, υπολογίζουμε μια προσέγγιση του μέγιστου δυνατού αριθμού σημείων του P που περιέχονται σε ένα ορθογώνιο παράλληλο στους άξονες με εμβαδόν ίσο με α. Γι’ αυτό το πρόβλημα χρησιμοποιούμε έναν 4–προσεγγιστικό αλγόριθμο που τρέχει σε χρόνο O(n log2 n).

**Λέξεις Κλειδιά:** Ορθογώνιο, Εμβαδόν, 4-προσεγγιστικός

Abstract

Let P be a set of n points in the plane. We show how to find, for a given integer k > 0, the smallest-area axis parallel rectangle that covers k points of P in O(n k2 log n + n log2 n) time. We also consider the problem of, given a value α > 0, covering as many points of P as possible with an axis-parallel rectangle of area at most α. For this problem we give a 4-approximation algorithm that works in O(n log2 n) time.

**Keywords:** Rectangle, Area, 4-approximation

Πίνακας περιεχομένων

Κεφάλαιο 1. Εισαγωγή1

1.1 Κύριοι Ορισμοί1

1.2 Αντικείμενο της Διπλωματικής Εργασίας2

1.3 Σχετικά Ερευνητικά Αποτελέσματα3

1.4 Δομή της Διπλωματικής Εργασίας4

Κεφάλαιο 2. Αλγόριθμοι Κάλυψης Σημείων5

2.1 Ελαχιστοποίηση Εμβαδού Ορθογωνίου για Δοθέν Πλήθος Σημείων5

2.2 Μεγιστοποίηση του Πλήθους Σημείων για Δοθέν Εμβαδόν Ορθογωνίου13

Κεφάλαιο 3. Η Υλοποίηση17

3.1 Είσοδος - Έξοδος17

3.2 Δομές Δεδομένων18

3.3 Ελαχιστοποίηση Εμβαδού Ορθογωνίου για Δοθέν Πλήθος Σημείων19

3.3.1 Η Συνάρτηση lemma1\_above19

3.3.2 Η Συνάρτηση binary\_search22

3.3.3 Η Συνάρτηση lemma1\_below23

3.3.4 Η Συνάρτηση lemma225

3.3.5 Η Συνάρτηση compute\_best\_top\_box27

3.3.6 Η Συνάρτηση compute\_best\_bot\_box28

3.3.7 Η Συνάρτηση lemma329

3.3.8 Η Συνάρτηση separate\_set30

3.3.9 Η Συνάρτηση find\_symmetric31

3.3.10 Η Συνάρτηση lemma432

3.3.11 Η Συνάρτηση minimize\_area33

3.3.12 Η Συνάρτηση find\_min34

3.3.13 Η Συνάρτηση plot\_points34

3.3.14 Παραδείγματα Εκτέλεσης του Προγράμματος35

3.4 Μεγιστοποίηση του Πλήθους Σημείων για Δοθέν Εμβαδόν Ορθογωνίου 39

3.4.1 Η Συνάρτηση max\_points\_in\_box39

3.4.2 Η Συνάρτηση current\_max\_inside40

3.4.3 Η Συνάρτηση find\_max42

3.4.4 Η Συνάρτηση plot\_points42

3.4.5 Παραδείγματα Εκτέλεσης του Προγράμματος42

Κεφάλαιο 4. Επίλογος45

Βιβλιογραφία46

# Εισαγωγή

## Κύριοι Ορισμοί

Σε αυτήν την ενότητα θα παρουσιάσουμε βασικές έννοιες και ορισμούς που είναι απαραίτητες για την περιγραφή και κατανόηση του προβλήματος που θα μας απασχολήσει σε αυτήν την εργασία. Πιο συγκεκριμένα θα αναλυθούν οι έννοιες του *πολυγώνου* και του *ορθογωνίου*.

Εικόνα που περιέχει κύκλος, σχεδίαση

Περιγραφή που δημιουργήθηκε αυτόματα Το *πολύγωνο* είναι ένα κλειστό επίπεδο γεωμετρικό σχήμα που αποτελείται από ευθύγραμμα τμήματα, τα οποία ενώνονται μεταξύ τους κατά τέτοιο τρόπο ώστε να σχηματίζουν μία κλειστή διαδρομή. Τα ευθύγραμμα αυτά τμήματα ονομάζονται πλευρές του πολυγώνου και τα σημεία στα οποία συναντώνται δύο πλευρές ονομάζονται κορυφές. Τα πολύγωνα μπορούν να έχουν διάφορες μορφές, ανάλογα με τον αριθμό των πλευρών τους. Επιπλέον, ένα πολύγωνο μπορεί να είναι κανονικό(όταν όλες οι πλευρές και όλες οι γωνίες είναι ίσες ή ακανόνιστο (όταν οι πλευρές ή οι γωνίες δεν είναι ίσες) (Σχήμα 1.1).

**Σχήμα 1.1:** Κανονικό πολύγωνο.

Εικόνα που περιέχει στιγμιότυπο οθόνης, ορθογώνιο παραλληλόγραμμο, τετράγωνο, σχεδίαση

Περιγραφή που δημιουργήθηκε αυτόματα Το *ορθογώνιο* είναι ένα τετράπλευρο γεωμετρικό σχήμα στο οποίο όλες οι γωνίες είναι ορθές. Ένα ορθογώνιο έχει δύο ζεύγη παράλληλων και ίσων πλευρών, όπου οι απέναντι πλευρές είναι ίσες μεταξύ τους. Το ορθογώνιο είναι μια ειδική περίπτωση παραλληλογράμμου (Σχήμα 1.2).

**Σχήμα 1.2:** Ορθογώνιο.

## Αντικείμενο της Διπλωματικής Εργασίας

Το αντικείμενο της Διπλωματικής Εργασίας είναι η μελέτη του προβλήματος της κάλυψης σημείων με ένα ορθογώνιο μικρού εμβαδού. Πιο συγκεκριμένα θα μελετήσουμε και θα υλοποιήσουμε τους αλγορίθμους των Berg, Cabello, Cheong, Eppstein και Knauer [6] για την επίλυση αυτού του προβλήματος. Στον πρώτο αλγόριθμο έχουμε ως στόχο την ελαχιστοποίηση του εμβαδού ενός ορθογωνίου που πρέπει να περικλείει συγκεκριμένο αριθμό σημείων. Ενώ στον δεύτερο αλγόριθμο, στόχος είναι η μεγιστοποίηση του πλήθους σημείων μέσα σε ένα ορθογώνιο δοθέντος του ακριβή εμβαδού του ορθογωνίου.

Η υλοποίηση πραγματοποιήθηκε στην γλώσσα προγραμματισμού Python, ενώ η οπτικοποίηση των δεδομένων και των πολυγώνων έγινε με χρήση της βιβλιοθήκης matplotlib της Python.

## Σχετικά Ερευνητικά Αποτελέσματα

Το πρόβλημα της κάλυψης σημείων με ένα ορθογώνιο μικρού εμβαδού έχει λάβει ιδιαίτερη προσοχή από πολλούς επιστήμονες. Αρχικά, οι Segal και Kedem [9] παρουσίασαν έναν O(n + k(n – k)2) αλγόριθμο όπου ο αριθμός των σημείων εντός του ορθογωνίου (k) προσεγγίζει τον αριθμό των σημείων (n) που περιλαμβάνει όλο το σετ P. Σε αντίθεση, εμείς μελετάμε την περίπτωση όπου το k είναι μικρό, έτσι ώστε να είναι προτιμότερο να μειώσουμε την εξάρτηση από το n και να αυξήσουμε την εξάρτηση από το k. Για την περίπτωση όπου το k έχει μικρή τιμή, αρκετές έρευνες [2, 4, 9] ισχυρίστηκαν ότι οι προηγούμενοι αλγόριθμοι των Aggarwal, Imai, Katoh και Suri [1] και των Eppstein και Errickson [7] λύνουν το πρόβλημα σε χρόνο O(k2 n log n) και O(n log n +k2n) αντίστοιχα. Ωστόσο, οι παραπάνω αλγόριθμοι εφαρμόζονται για την εκδοχή ελάχιστης περιμέτρου του προβλήματος. Δεν λειτουργούν για την εκδοχή του ελάχιστου εμβαδού, διότι βασίζονται στο γεγονός ότι, στην περίπτωση ελάχιστης περιμέτρου, το βέλτιστο υποσύνολο των k σημείων, το υπολογίζουμε από τους O(k) κοντινότερους γείτονες ενός σημείου, κάτι που δεν αληθεύει για την περίπτωση ελάχιστου εμβαδού. Το ίδιο πρόβλημα αντιμετωπίζουμε και όταν προσπαθούμε να υλοποιήσουμε τους αλγορίθμους των Datta, Lenhof, Schwarz και Smid [5]. Για την επίλυση του προβλήματός μας, δε μπορούμε να σταθούμε μόνον στα σύνολα τον κοντινότερων γειτόνων, αλλά πρέπει να χρησιμοποιήσουμε διαφορετικές μεθόδους για να πετύχουμε τους αλγοριθμικούς χρόνους που θέλουμε.

Αργότερα, οι Kaplan, Roy και Sharir [8] έδειξαν ότι το πρόβλημά μας μπορεί να λυθεί σε χρόνο O(n5/2 log2n). Αυτός είναι και ο πρώτος υπό-κυβικός αλγόριθμος που είναι και αρκετά αποτελεσματικός, σε σχέση με προηγούμενους αλγόριθμους, για διάφορες τιμές του k.

Έχουν υπάρξει και αρκετές προσπάθειες ερευνητών για το πρόβλημα της ελαχιστοποίησης του μεγέθους δίσκου που περιέχει k σημεία. Σε αυτό το πρόβλημα έχουν υπάρξει πολύ αποδοτικές λύσεις διότι δεν έχει σημασία εάν θα ελαχιστοποιήσουμε την περίμετρο ή το εμβαδό του δίσκου.

Όσον αφορά τους αλγορίθμους που υλοποιήσαμε, υπολογίζουμε το ελάχιστο εμβαδό που περικλείει k σημεία του συνόλου P σε χρόνο O(nk2logn + nlog2n) χρησιμοποιώντας κατάλληλα τη μέθοδο διαίρει-και-βασίλευε που μοιάζει με αυτή που χρησιμοποίησαν οι Aronov, Ezra και Sharir [3]. Αυτός είναι και ο μόνος γνωστός αλγόριθμος με σχεδόν γραμμική εξάρτηση από το n. Για τον υπολογισμό του μέγιστου πλήθους σημείων που μπορούν να περικλείονται σ’ ένα ορθογώνιο με συγκεκριμένο εμβαδό κατασκευάσαμε έναν 4-προσεγγιστικό O(n log2n) αλγόριθμο, που χρησιμοποιεί με παρόμοιο τρόπο την τεχνική διαίρει-και-βασίλευε.

## Δομή της Διπλωματικής Εργασίας

Η Διπλωματική Εργασία αποτελείται από τέσσερα κεφάλαια. Το Κεφάλαιο 1 αποτελεί την εισαγωγή, όπου γίνεται αναφορά σε βασικές έννοιες και ορισμούς που έχουν σχέση με τα πολύγωνα και τις ιδιότητές τους. Στη συνέχεια παρουσιάζεται το αντικείμενο της Διπλωματικής Εργασίας και σχετικά ερευνητικά αποτελέσματα.

Στο Κεφάλαιο 2 περιγράφουμε αναλυτικά τους αλγορίθμους ελαχιστοποίησης εμβαδού ορθογωνίου για δοθέν πλήθος σημείων και μεγιστοποίησης του πλήθους σημείων για δοθέν εμβαδόν ορθογωνίου των Berg, Cabello, Cheong, Eppstein και Knauer [6] παρουσιάζοντας το θεωρητικό υπόβαθρο, την πολυπλοκότητα αλλά και τα βήματα εκτέλεσης των αλγορίθμων μέσω ενός απλού παραδείγματος.

Στο Κεφάλαιο 3 παρουσιάζεται η ακριβής υλοποίηση των αλγορίθμων στην γλώσσα προγραμματισμού Python, περιγράφοντας όλες τις συναρτήσεις αυτών. Επίσης, περιγράφονται η μορφή της εισόδου και της εξόδου του προγράμματος καθώς και οι δομές δεδομένων που χρησιμοποιήθηκαν. Τέλος, υπάρχουν κάποια παραδείγματα εκτέλεσης των προγραμμάτων.

Στο Κεφάλαιο 4 συνοψίζουμε τα αποτελέσματα της εργασίας και παρουσιάζουμε ενδιαφέρουσες επεκτάσεις του προβλήματος.

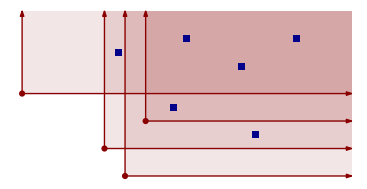
# Αλγόριθμοι Κάλυψης Σημείων

## Ελαχιστοποίηση Εμβαδού Ορθογωνίου για Δοθέν Πλήθος Σημείων

Αρχικά, θεωρούμε ένα σύνολο P των n σημείων στο χώρο R2 καθώς και έναν ακέραιο k 2, που είναι τα σημεία που θέλουμε να περιέχονται στο ορθογώνιο. Επίσης, για ένα ορθογώνιο R συμβολίζουμε με top(R) και bot(R) την υψηλότερη και χαμηλότερη ακμή του αντίστοιχα. Για ένα σημείο p R2, χρησιμοποιούμε τα σύμβολα px και py για τις x- και y-συντεταγμένες, αντίστοιχα.

Ακόμη, το ελάχιστο εμβαδό που ζητείται συμβολίζεται ως

area\*(P, k) = min { area(R) | R is a box with | R P| k }.

Για την ευκολότερη κατανόηση και υλοποίηση του αλγορίθμου, η λύση του παρουσιάζεται τμηματικά σε λήμματα, όπως περιγράφεται από τους Berg, Cabello, Cheong, Eppstein και Knauer [6].

**Σχήμα 2.1:** ορθογώνιο δύο πλευρών που χρησιμοποιείται στο Λήμμα 1[6].

**Λήμμα 1.** *Θεωρούμε δύο σύνολα Α και Β με τουλάχιστον n σημεία στο χώρο R2.*

*Για κάθε σημείο b B, ορίζουμε ως Rb το ορθογώνιο δύο πλευρών [bx, [by, ), όπως φαίνεται και στο παραπάνω σχήμα (Σχήμα 2.1). Σε χρόνο O(kn + nlogn) μπορούμε να βρούμε, για όλα τα b B, τα k σημεία στο A Rb με τη μικρότερη x-συντεταγμένη.*

*Απόδειξη.* Μπορούμε να λύσουμε το πρόβλημα αυτό χρησιμοποιώντας έναν αλγόριθμο με γραμμή-σάρωσης, που σαρώνει το χώρο, από αριστερά προς τα δεξιά, με μια κάθετη ευθεία l. Παρακάτω δίνονται περισσότερες λεπτομέρειες.

Ορίζουμε ως Al και Bl τα σημεία που βρίσκονται αριστερά της ευθείας l και ανήκουν στα σύνολα A και B, αντίστοιχα. Θεωρούμε την οικογένεια ορθογωνίων Rl = { Rb | b Bl }. Ανά πάσα στιγμή, διατηρούμε το υποσύνολο Rl’ του συνόλου Rl, που αποτελείται από ορθογώνια που δεν περιέχουν k σημεία του Al. Τα ορθογώνια Rb  Rl’ αποθηκεύονται σε ένα ισορροπημένο δέντρο δυαδικής αναζήτησης T, το οποίο είναι ταξινομημένο με βάση την τιμή by. Επιπλέον, για κάθε ορθογώνιο Rb Rl’, αποθηκεύουμε σε μία λίστα Lb τα σημεία του Al που περιέχει καθώς επίσης και το μέγεθος της λίστας Lb, που το παίρνουμε υπολογίζοντας την τιμή | Rb Al|.

Όταν η γραμμή l συναντήσει σημείο α A, υπολογίζουμε τα m ορθογώνια του Rl’ που περιέχουν το α χρησιμοποιώντας το δέντρο T, αυτό γίνεται σε χρόνο O(m + logn). Για κάθε ένα από τα m ορθογώνια του Rb Rl’ που περιέχουν το α, προσθέτουμε το α στη λίστα Lb. Επιπλέον, εάν η λίστα Lb περιέχει k σημεία, τότε το Rb δεν ανήκει πλέον στο Rl’ και έτσι αφαιρούμε την εγγραφή από το δέντρο T.

Όταν η γραμμή συναντήσει σημείο b B, τότε το Rb γίνεται μέλος του συνόλου Rl και το εισάγουμε στο T. Αν υπάρχει σημείο α που ανήκει και στο A και στο B, πρώτα θεωρούμε το α ως στοιχείο του συνόλου B και έπειτα του A. Έτσι, το α γίνεται στοιχείο του Rb.

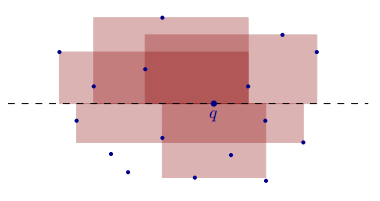
Κάθε εισαγωγή ή διαγραφή στο T παίρνει O(logn). Κάνουμε |B| εισαγωγές και τουλάχιστον |B| διαγραφές στο T, σε συνολικό χρόνο O(nlogn). Για κάθε σημείο α A, ξοδεύουμε χρόνο O(logn) συν Ο(1) για κάθε ορθογώνιο Rb το οποίο περιέχει το α. Ο συνολικός χρόνος εκτέλεσης είναι O(kn + nlogn).

Ο παρακάτω ορισμός θα μας χρειαστεί στο Λήμμα 2:

Για ένα σύνολο σημείων Q, ένα σημείο q Q, και μια παράμετρο k, ορίζουμε

Φ(Q, q, k) := min { εμβαδόν(R) | όπου το R είναι ένα ορθογώνιο με το q top(R) ή q bot(R) και το R περιέχει τουλάχιστον k σημεία του Q }

Στο σχήμα που ακολουθεί (Σχήμα 2.2) φαίνεται ένα παράδειγμα για καλύτερη κατανόηση του παραπάνω ορισμού.

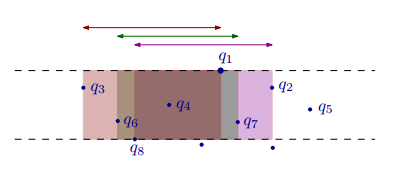


**Σχήμα 2.2:** διάφορα ορθογώνια που υπολογίζονται κατά τον υπολογισμό της συνάρτησης Φ(Q, q, k) για k=5[6].

**Λήμμα 2.** *Δοθέντος των Q, q και k, μπορούμε να υπολογίσουμε το Φ(Q, q, k) σε χρόνο O(|Q|2).*

*Απόδειξη.* Ας παρουσιάσουμε την περίπτωση όπου το q top(R), η άλλη περίπτωση είναι παρόμοια. Έστω ότι q1, q2,…, qm είναι τα σημεία του Q για τα οποία η y-συντεταγμένη δεν είναι μεγαλύτερη από qy, σε φθίνουσα σειρά με βάση την y-συντεταγμένη. Επίσης, θεωρούμε Qi = {q1,…, qi}.

Ταξινομούμε τα στοιχεία του Qi σε αύξουσα σειρά με βάση την x-συντεταγμένη, τότε μπορούμε να βρούμε σε χρόνο O(|Qi|) = O(i) το ορθογώνιο R με το ελάχιστο εμβαδόν που περιέχει k σημεία, δεδομένου ότι q top(R) και qi bot(R), χρησιμοποιώντας μια γραμμική αναζήτηση στη λίστα με δύο δείκτες που απέχουν μεταξύ τους k σημεία. Ένα παράδειγμα φαίνεται στο Σχήμα 2.3.

 Στη συνέχεια ακολουθούμε την εξής διαδικασία. Πρώτα υπολογίζουμε τα σετ Qm και τα ταξινομούμε με βάση την x-συντεταγμένη σε χρόνο O(|Q|log|Q|). Έπειτα, συνεχώς υπολογίζουμε το ορθογώνιο με το μικρότερο εμβαδόν για το τρέχον σετ Qi (αρχικά i = m) σε χρόνο O(i), μετά αφαιρούμε από την λίστα το στοιχείο με την μικρότερη y-συντεταγμένη για να πάρουμε το σύνολο Qi-1, σε χρόνο O(i). Συνεπώς ο συνολικός χρόνος εκτέλεσης είναι O(|Q|2).

**Σχήμα 2.3:** τα ορθογώνια όταν k=5, q top(R) και Q8 = {q1,…, q8}[6].

Ο παρακάτω ορισμός θα μας βοηθήσει στην απόδειξη του Λήμματος 3:

Για ένα σύνολο σημείων P, μια οριζόντια ευθεία l και μια παράμετρο k, ορίζουμε

Ψ(P, l, k) := min { εμβαδόν(R) | όπου R ορθογώνιο που τέμνει την ευθεία l έτσι ώστε το R να περιέχει τουλάχιστον k σημεία του P }

Αφού εξ ορισμού η τιμή area\*(P, k), είναι το εμβαδόν της βέλτιστης λύσης για το πρόβλημά μας, είναι προφανές ότι area\*(P, k) Ψ(P, l, k). Το παρακάτω λήμμα εξηγεί ότι όταν το ορθογώνιο μιας βέλτιστης λύσης τέμνεται από την ευθεία l, τότε μπορούμε να μειώσουμε την αναζήτησή μας σε μικρότερα προβλήματα μεγέθους O(k).

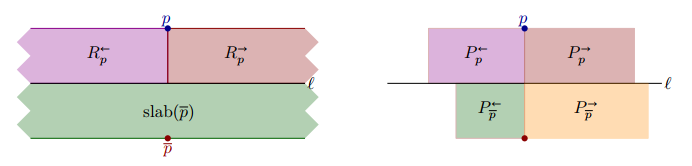
**Λήμμα 3.** *Δοθέντος των P, l και k, μπορούμε να υπολογίσουμε σε χρόνο O(kn + nlogn) υποσύνολα του P, Qp, που δεικτοδοτούνται από το p P, με τις εξής ιδιότητες:*

* *Το Qp έχει O(k) σημεία για κάθε p P.*
* *Για κάθε k’ k, εάν area\*(P, k) = Ψ(P, l, k), τότε area\*(P, k) = Φ(Qp, p, k’) για κάποιο p P.*

*Απόδειξη.* Για κάθε p P, ορίζουμε ως p’ το συμμετρικό του p ως προς την ευθεία l. Για κάθε σημείο q του χώρου, όπου q ∉ l, ορίζουμε τα παρακάτω:

* Ορίζουμε ως slab(q) το χώρο μεταξύ της l και της παράλληλης ευθείας ως προς την l που διέρχεται από το σημείο q.
* Ορίζουμε ως Rq→ το ορθογώνιο τριών πλευρών slab(q) { (x,y) R2 | x qx } και επιπλέον ορίζουμε ως Pq→ τα k σημεία του P με την μικρότερη χ-συντεταγμένη μέσα στο Rq→.
* Ορίζουμε ως Rq← το ορθογώνιο τριών πλευρών slab(q) { (x,y) R2 | x qx } και επιπλέον ορίζουμε ως Pq← τα k σημεία του P με την μεγαλύτερη χ-συντεταγμένη μέσα στο Rq←.

Για καλύτερη κατανόηση αυτών τον όρων δείχνουμε το Σχήμα 2.4.



**Σχήμα 2.4:** η ορολογία που χρησιμοποιούμε στο Λήμμα 3[6].

Για κάθε p P, ορίζουμε ως Qp την ένωση των Pp→, Pp←, Pp’→ και Pp’←. Είναι ξεκάθαρο ότι κάθε σετ Qp έχει το πολύ 4k σημεία, επομένως η πρώτη ιδιότητα του λήμματος ευσταθεί.

Για να αποδείξουμε την δεύτερη ιδιότητα θεωρούμε κάποια τιμή k’ k και υποθέτουμε ότι area\*(P, k’) = Ψ(P, l, k’). Αυτό συνεπάγεται ότι υπάρχει ένα ορθογώνιο R\* με το ελάχιστο εμβαδόν όπου area(R\*) = area\*(P, k’) έτσι ώστε το R\* να τέμνει την l. Ορίζουμε ως t\* και b\* να είναι σημεία του P, όπου ανήκουν στο top(R\*) και bot(R\*) αντίστοιχα. Υποθέτουμε ότι η απόσταση από το t\* στο l είναι τουλάχιστον όσο αυτή από το b\* στο l. Αυτό σημαίνει ότι το R\* P περιέχεται στο slab(t\*) slab(t\*’).

Έπειτα, αποδεικνύουμε ότι το R\* P περιέχεται στο Qt\*. Θα το αποδείξουμε με απαγωγή σε άτοπο. Έστω ότι το R\* P περιέχει σημείο α που δεν ανήκει στο Qt\*. Για διευκόλυνση παρακάτω υπάρχει και το Σχήμα 2.5. Επομένως, το σημείο α περιέχεται σε κάποιο από τα ορθογώνια τριών πλευρών που χρησιμοποιούνται για τον ορισμό του Qt\*, δηλαδή σε κάποιο από τα Rt\*→, Rt\*←, Rt\*’→ και Rt\*’←. Έστω ότι R’’ { Rt\*→, Rt\*←, Rt\*’→, Rt\*’← } είναι το ορθογώνιο τριών πλευρών που περιέχει το σημείο α, P’’ { Pt\*→, Pt\*←, Pt\*’→, Pt\*’← } είναι τα σύνολα σημείων που περιέχονται στο R’’ και τέλος q’’ είναι σημείο του P’’ και είναι αυτό που έχει τη μεγαλύτερη απόσταση από την κάθετη ευθεία που διατρέχει το t\*. Με τον τρόπο που επιλέξαμε τα σημεία του P’’ μέσα στο R’’ έχουμε

|tx\* - qx’’| < |tx\* - αx|.

Να σημειωθεί ότι το πλαίσιο (bounding box) bb(P’’) του P’’ περιέχει k k’ σημεία και έχει μέγιστο δυνατό εμβαδόν

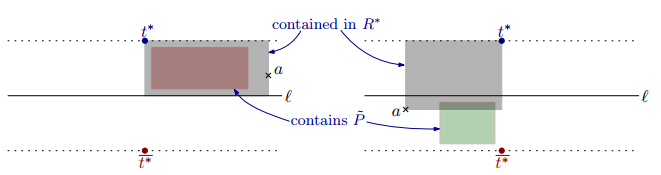
|tx\* - qx’’| dist(t\*, l)

όπου dist(t\*, l) είναι η κάθετη απόσταση από το t\* στην ευθεία l. Από την άλλη πλευρά, αφού το R\* τέμνει την l και έχει τα σημεία α και t\* στα άνω και κάτω άκρα του, ισχύει ότι

area(R\*) |tx\* - αx| dist(t\*, l) > |tx\* - qx’’| dist(t\*, l) area(bb(P’’)).

Το παραπάνω όμως αντιτίθεται στο γεγονός ότι το εμβαδόν του R\* είναι το βέλτιστο που μπορεί να καλύπτει k’ σημεία. Έτσι, αποδείξαμε ότι το R\* P περιέχεται στο Qt\*, και επομένως ισχύει και η δεύτερη ιδιότητα του λήμματος.

Τέλος, απομένει να αποδείξουμε ότι η κατασκευή των συνόλων Qp, για κάθε p P, μπορεί να γίνει σε χρόνο O(kn + nlogn). Για να πετύχουμε το παραπάνω χρησιμοποιούμε το Λήμμα 1 μερικές φορές. Έστω ότι P+ και P- είναι τα σημεία πάνω και κάτω από την ευθεία l αντίστοιχα. Επίσης, ορίζουμε P+’ = { p’ | p P+ } και P-΄ = { p’ | p P- }. Τα σύνολα σημείων Pq→για κάθε q P- P+’, υπολογίζονται χρησιμοποιώντας το Λήμμα 1 για A = P- και B = P- P+’. Τα σύνολα Pq→ για q P+ P-’, Pq← για q P- P+’ και Pq← για q P+ P-’, υπολογίζονται με παρόμοιο τρόπο χρησιμοποιώντας συμμετρικές εκδοχές του Λήμματος 1.



**Σχήμα 2.5:** μέρος της απόδειξης του Λήμματος 3[6].

**Λήμμα 4.** *Έστω P* *σύνολο των n σημείων, μία οριζόντια ευθεία l και k θετικός ακέραιος. Δαπανώντας χρόνο προετοιμασίας O(nk + nlogn), υπολογίζουμε σε χρόνο O(nk2), για οποιονδήποτε k’ k, μια τιμή V(P, l, k’) με τις εξής ιδιότητες:*

* *area\*(P, k’) V(P, l, k’),*
* *εάν area\*(P, k’) = Ψ(P, l, k’), τότε V(P, l, k’) = area\*(P, k’).*

*Απόδειξη.* Υπολογίζουμε τα σύνολα Qp, δεικτοδοτούμενα με p P, με τον τρόπο που δείξαμε στο Λήμμα 3. Έτσι, ο χρόνος προετοιμασίας των συνόλων είναι O(nk + nlogn).

Υποθέτουμε ότι μας δίνεται τιμή k’ k.Για κάθε p P, χρησιμοποιούμε το Λήμμα 2 για να υπολογίσουμε την τιμή Φ(Qp, p, k’) σε χρόνο O(|Qp|2) = O(k2). Έπειτα, επιστρέφουμε την τιμή V(P, l, k’) = min { Φ(Qp, p, k’) | p P }. Ο υπολογισμός παίρνει χρόνο O(nk2).

Αφού για κάθε p P η τιμή Φ(Qp, p, k’) είναι το εμβαδόν ενός ορθογωνίου που περιέχει k’ σημεία του P, ισχύει ότι V(P, l, k’) area\*(P, k’). Αν area\*(P, k’) = Ψ(P, l, k’), τότε το Λήμμα 3 μας εγγυάται ότι area\*(P, k’) = Φ(Qp0, p0, k’) για κάποιο p0 P και συνεπώς

area\*(P, k’) = Φ(Qp0, p0, k’) min { Φ(Qp, p, k’) | p P } = V(P, l, k’).

Συμπεραίνουμε ότι V(P, l, k’) = area\*(P, k’).

**Θεώρημα 5.** *Δοθέντος ενός συνόλου των n σημείων P και μιας τιμής k, μπορούμε να προ-επεξεργαστούμε το P έτσι ώστε σε χρόνο O(nklogn + nlog2n), για οποιοδήποτε k’ k, μπορούμε να βρούμε σε O(nk2logn) ένα ορθογώνιο ελάχιστου εμβαδού που να περιέχει τουλάχιστον k’ σημεία του P.*

*Απόδειξη.* Θεωρούμε μια οριζόντια ευθεία l που να χωρίζει το σύνολο P σε δύο μικρότερα υποσύνολά του P+ και P-, αναλόγως αν βρίσκονται πάνω ή κάτω από την ευθεία l αντίστοιχα. Για οποιονδήποτε αριθμό k’, όπου 1 k’ n έχουμε

area\*(P, k’) = min { Ψ(P, l, k’), area\*(P+, k’), area\*(P-, k’) }

Πράγματι, μία βέλτιστη λύση που περιέχει k’ σημεία είτε τέμνεται από την l είτε περιέχει σημεία ενός εκ των δύο συνόλων P+ και P-. Αυτή είναι και η βάση δημιουργίας ενός αλγορίθμου που βασίζεται στην τεχνική διαίρει-και-βασίλευε.

Στο βήμα της προ-επεξεργασίας, χρησιμοποιούμε το Λήμμα 4 για P, l και k, και παίρνει χρόνο O(nk + nlogn) και έπειτα αναδρομικά προ-επεξεργαζόμαστε τα P+ και P-. Αφού η αναδρομή έχει logn στάδια και δύο σύνολα στο ίδιο επίπεδο είναι ξένα μεταξύ τους, καταναλώνουμε χρόνο ίσο με O(nklogn + nlog2n) στην προ-επεξεργασία.

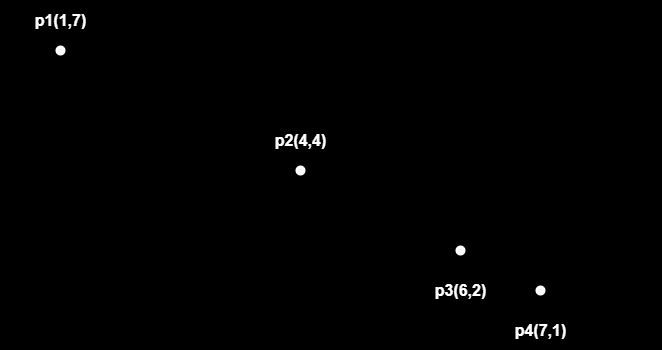
Δοθέντος μιας τιμής k’, υπολογίζουμε την τιμή area\*(P, k’) όπως εξηγείται παρακάτω. Στο πρώτο επίπεδο της αναδρομής, για το σύνολο P και την ευθεία l, υπολογίζουμε την τιμή V(P, l, k’) σε χρόνο O(nk2), χρησιμοποιώντας το Λήμμα 4. Στη συνέχεια, αναδρομικά υπολογίζουμε τις τιμές area\*(P+, k’) και area\*(P-, k’). Τελικά, επιστρέφουμε την τιμή

area\*(P, k’) = min { V(P, l, k’), area\*(P+, k’), area\*(P-, k’) }.

Γνωρίζοντας ότι έχουμε logn αναδρομικά επίπεδα, ο αλγοριθμός μας τρέχει σε χρόνο O(nk2logn).

**Συμπέρασμα 6.** *Δοθέντος ενός συνόλου n σημείων P και μιας τιμής k μπορούμε να βρούμε σε χρόνο O(nk2logn + nlog2n) ένα ορθογώνιο ελάχιστου εμβαδού που να περιέχει τουλάχιστον k σημεία του P.*

*Απόδειξη.* Εφαρμόζουμε το Θεώρημα 5 για k’ = k. Σε αυτήν την περίπτωση μπορούμε να απορρίψουμε το βήμα της προ-επεξεργασίας και σε κάθε επίπεδο της αναδρομής, υπολογίζουμε την τιμή Φ(Qp, p, k) αμέσως μετά τον υπολογισμό των συνόλων Qp.

 Παρακάτω ακολουθεί ένα απλό παράδειγμα εκτέλεσης του αλγορίθμου για k = 3. Θεωρούμε ένα σύνολο P που αποτελείται από τέσσερα σημεία στον χώρο όπως φαίνεται στο Σχήμα 2.6.

**Σχήμα 2.6**

Αρχικά, χωρίζουμε το σύνολο των σημείων P, σε δύο υποσύνολα P+ και P-, με βάση το αν βρίσκονται πάνω ή κάτω από μια οριζόντια ευθεία l, αντίστοιχα (Σχήμα 2.7). Αυτή η ευθεία υπολογίζεται ως

l = (yp2 + yp3) / 2 = 3

Εικόνα που περιέχει στιγμιότυπο οθόνης, κείμενο, χώρος, αστρονομία

Περιγραφή που δημιουργήθηκε αυτόματαΣτο Σχήμα 2.7 φαίνονται και τα συμμετρικά των σημείων ως προς την ευθεία l. Τα υπολογίζουμε γιατί τα χρειαζόμαστε στην κατασκευή των συνόλων Qp.

**Σχήμα 2.7**

Έπειτα, υπολογίζουμε την τιμή V(P, l=3, k=3) = min { Φ(Qp, p, k’) | p P }. Επομένως, υπολογίζουμε τα σύνολα Qp ως εξής:

Pp1→ = [p1, p2], Pp1← = [p1], Pp1’→ = [p3, p4], Pp1’← = []. Άρα, Qp1 = [p1, p2, p3, p4].

Pp2→ = [p2], Pp2← = [p2], Pp2’→ = [p3], Pp2’← = []. Άρα, Qp2 = [p2, p3].

Pp3→ = [p3], Pp3← = [p3], Pp3’→ = [], Pp3’← = [p2]. Άρα, Qp3 = [p2, p3].

Pp4→ = [p4], Pp4← = [p4, p3], Pp4’→ = [], Pp4’← = [p2]. Άρα, Qp4 = [p2, p3, p4].

Τα σύνολα Qp2 και Qp3, δεν μπορούν να μας δώσουν κάποια πιθανή λύση γιατί περιέχουν δύο σημεία που είναι λιγότερα του k = 3.

Στη συνέχεια υπολογίζουμε τις τιμές Φ(Qp, p, k) για τα σύνολα Qp1 και Qp4:

* Φ(Qp1, p1, k=3) = (xp3 – xp1) (yp1 – yp3) = 5 5 = 25

Το εμβαδόν το παίρνουμε αφού έχουμε αφαιρέσει από το Qp1 το p4, που έχει την μικρότερη y-συντεταγμένη, για να έχουμε ορθογώνιο που καλύπτει τρία σημεία.

* Φ(Qp4, p4, k=3) = (xp4 – xp2) (yp2 – yp1) = 3 3 = 9.

Τελικά, έχουμε V(P, l=3, k=3) = min { Φ(Qp1, p1, k=3), Φ(Qp4, p4, k=3) } = 9 και το ορθογώνιο που προκύπτει ορίζεται ως

[xlow, xhigh, ylow, yhigh] = [4, 7, 1, 4].

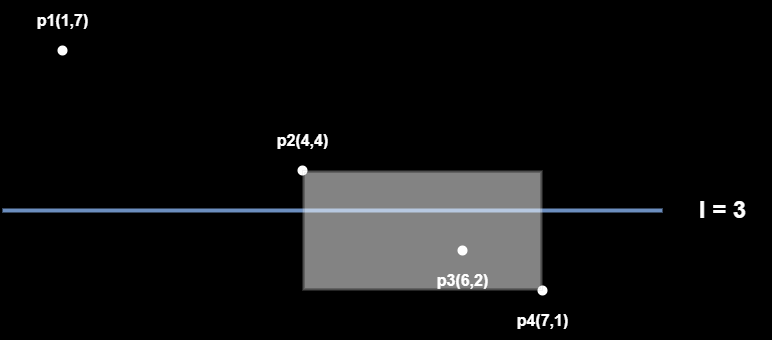
Ακολούθως, συνεχίζεται η εκτέλεση του αλγορίθμου για τα δύο υποσύνολα P+ και P-. Όμως, και τα δύο αυτά υποσύνολα, περιέχουν δύο μόνο σημεία που είναι λιγότερα των σημείων που ζητάμε να περιέχει το ορθογώνιο (k = 3). Συνεπώς, επιστρέφουμε μία πολύ μεγάλη τιμή (+).

Τελικά, επιστρέφεται

area\*(P, k) = min { V(P, l, k), area\*(P+, k), area\*(P-, k) }

= min { 9, +, + }

= 9.

**Επομένως, το ζητούμενο ορθογώνιο είναι αυτό που σημειώνεται στο Σχήμα 2.8.

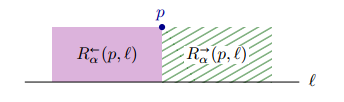
**Σχήμα 2.8**

## Μεγιστοποίηση του Πλήθους Σημείων για Δοθέν Εμβαδόν Ορθογωνίου

Σε αυτήν την περίπτωση εξετάζουμε το πρόβλημα εύρεσης του μέγιστου αριθμού σημείων που μπορούν να καλυφθούν από ορθογώνιο εμβαδού α > 0. Συμβολίζουμε το ζητούμενο ως κ\*(P, α). Ο αλγόριθμος που θα εξετάσουμε είναι ένας 4-προσεγγιστικός αλγόριθμος, δηλαδή έστω κ(P, α) η τιμή που επιστρέφει ο αλγόριθμός μας, τότε γνωρίζουμε και θα αποδείξουμε ότι η τιμή ικανοποιεί την παρακάτω συνθήκη

κ\*(P, α)/4 κ(P, α) κ\*(P, α)

Για μια οριζόντια ευθεία l και ένα σημείο p ∉ l, ορίζουμε ως Rα→(p, l) το ορθογώνιο που έχει εμβαδόν α, το p ανήκει σε μία από τις γωνίες του, έχει ακμή που περιέχεται στην l και περιέχει σημεία με x-συντεταγμένη μεγαλύτερη από px. Παρόμοια, ορίζουμε το ορθογώνιο Rα←(p, l), με τη διαφορά ότι περιέχει σημεία με x-συντεταγμένη μικρότερη από px. Για μεγαλύτερη πληρότητα δίνεται το Σχήμα 2.6. Επίσης, Rα(l) είναι το σύνολο των ορθογωνίων p P { Rα→(p, l), Rα←(p, l) }. Τέλος, έστω κ\*(P, l, α) είναι ο μέγιστος αριθμός σημείων του P που καλύπτονται από ορθογώνιο εμβαδού α και τέμνει την ευθεία l.



**Σχήμα 2.9:** τα ορθογώνια Rα←(p, l) και Rα→(p, l)[6].

**Λήμμα 7.**  *Υπάρχει κάποιο R Rα(l) τέτοιο ώστε |P R| κ\*(P, l, α)/4.*

*Απόδειξη.* Έστω R\* ορθογώνιο εμβαδού α που περιέχει κ\*(P, l, α) σημεία και τέμνει την l. Επίσης, l+ και l- αναπαριστούν τα ημι-επίπεδα πάνω και κάτω από την l αντίστοιχα. Ορίζουμε ως R+ := R\* l+ και R- := R\* l- και υποθέτουμε, χωρίς απώλεια της γενικότητας, ότι |R+  P| |R\* P|/2. Ως p συμβολίζουμε το σημείο στο P R\* με την μέγιστη y-συντεταγμένη. Τότε R+ ⊂ Rα→(p, l) Rα←(p, l). Συνεπώς, τουλάχιστον ένα από τα Rα→(p, l), Rα←(p, l) πρέπει να περιέχει

|R+  P|/2 |R\* P|/2 σημεία.

**Θεώρημα 8.** *Δοθέντος ενός συνόλου n σημείων P και μιας τιμής α > 0, μπορούμε να υπολογίσουμε σε χρόνο O(nlog2n) τιμή κ(P, α) για την οποία ισχύει*

*κ\*(P, α)/4 κ(P, α) κ\*(P, α).*

*Απόδειξη.* Προ-επεξεργαζόμαστε το P για ερωτήματα μέτρησης ορθογωνίων με βάση την τεχνική που περιγράφεται από τον Willard [10]. Αυτό γίνεται σε χρόνο O(nlogn) και για κάθε ερώτημα ορθογωνίου R μπορούμε να αναφέρουμε την τιμή |R P| σε χρόνο O(logn).

Έπειτα, αναπτύσσουμε έναν αναδρομικό αλγόριθμο. Θεωρούμε μια ευθεία l που χωρίζει το P σε δύο σύνολα P+ και P- του ίδιου μεγέθους. Ισχύει ότι

κ\*(P, α) = max { κ\*(P+, α), κ\*(P-, α), κ\*(P, l, α) }

Δημιουργούμε το σύνολο Rα(l) σε χρόνο O(n). Για κάθε ορθογώνιο R Rα(l) κάνουμε αναζήτηση στην δομή δεδομένων για να βρούμε την τιμή |R P|. Στη συνέχεια, υπολογίζουμε την τιμή κ(P, l, α) = max { |R P| | R Rα(l) } σε χρόνο O(nlogn). Από το Λήμμα 7 έχουμε κ\*(P, l, α)/4 κ(P, l, α) κ\*(P, l, α). Τότε, επιστρέφουμε την μέγιστη τιμή εκ των κ(P, l, α) και κ(P+, α), κ(P-, α) που υπολογίζονται αναδρομικά από τα σύνολα P+ και P- αντίστοιχα. Αφού σε κάθε επίπεδο της αναδρομής τα σύνολα είναι ξένα μεταξύ τους, σε κάθε επίπεδο της αναδρομής καταναλώνουμε χρόνο O(nlogn) για συνολικό χρόνο O(nlog2n).

Αφού ο αλγόριθμος τρέχει για ορθογώνια εμβαδού α, η μέγιστη δυνατή επιστρεφόμενη τιμή είναι κ\*(P, α). Λαμβάνοντας υπόψιν ότι κ(P+, α) κ\*(P+, α)/4, κ(P-, α) κ\*(P-, α)/4 και κ(P, l, α) κ\*(P, l, α)/4 προκύπτει ότι

κ(P, α) = max { κ(P+, α), κ(P-, α), κ(P, l, α) }

max { κ\*(P+, α)/4, κ\*(P-, α)/4, κ\*(P, l, α)/4 }

= κ\*(P, α)/4.

Εικόνα που περιέχει στιγμιότυπο οθόνης, μαύρο, κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα Παρακάτω δίνονται τα βήματα εκτέλεσης του αλγορίθμου για ένα απλό παράδειγμα για εμβαδόν α = 10 και σύνολο P των τεσσάρων σημείων όπως φαίνεται στο Σχήμα 2.10.

**Σχήμα 2.10**

Εν συνεχεία, υπολογίζουμε το σύνολο των σημείων Rα(l), με βάση την ευθεία l = 3 που χωρίζει το P σε P+ και P-, και φαίνεται στο Σχήμα 2.11. Να σημειωθεί ότι κάθε ορθογώνιο του Rα(l) πρέπει να έχει εμβαδό α = 10, που είναι και το δοθέν εμβαδόν ορθογωνίου. Θα συμβολίζουμε ένα ορθογώνιο με την μορφή [xlow, xhigh, ylow, yhigh].

* Για το σημείο p1:

Βρίσκουμε το Rα→(p1, l=3), για το οποίο δε γνωρίζουμε το xhigh. Το υπολογίζουμε χρησιμοποιώντας την εξίσωση α = β υ, έτσι προκύπτει xhigh = 3.5 και προκύπτει το ορθογώνιο [1, 3.5, 3, 7]. Άρα, |Rα→(p1, l=3) P| = 1. Παρόμοια προκύπτει ότι |Rα←(p1, l=3) P| = 1.

* Για το σημείο p2:

Με παρόμοιο τρόπο υπολογίζουμε τα Rα→(p2, l=3) και Rα←(p2, l=3). Προκύπτει |Rα→(p2, l=3) P| = 1 και |Rα←(p2, l=3) P| = 1.

* Για το σημείο p3:

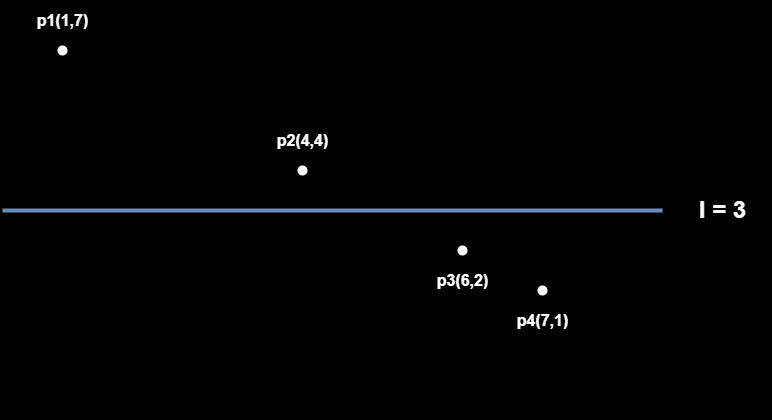
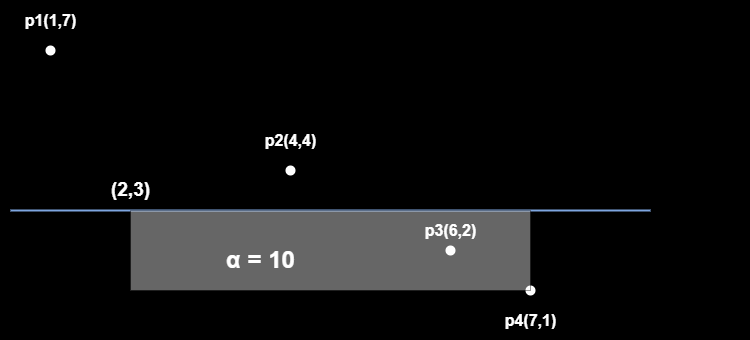
Με παρόμοιο τρόπο υπολογίζουμε τα Rα→(p3, l=3) και Rα←(p3, l=3). Προκύπτει |Rα→(p3, l=3) P| = 1 και |Rα←(p3, l=3) P| = 1.

* Για το σημείο p4:

Με παρόμοιο τρόπο υπολογίζουμε τα Rα→(p4, l=3) και Rα←(p4, l=3). Προκύπτει |Rα→(p4, l=3) P| = 1 και |Rα←(p4, l=3) P| = 2. Το ορθογώνιο που καλύπτει δύο σημεία είναι το [xlow, xhigh, ylow, yhigh] = [2, 7, 1, 3].

Συνεπώς, έχουμε κ(P, l=3, α=10) = 2 σημεία.

Παρόμοια για τα σύνολα P+ και P- δεν επιστρέφεται ορθογώνιο που να περιέχει περισσότερα από ένα σημεία. Τελικά, το ζητούμενο ορθογώνιο (Σχήμα 2.12) εμβαδού α=10 είναι το

[xlow, xhigh, ylow, yhigh] = [2, 7, 1, 3].

**Σχήμα 2.12**

**Σχήμα 2.11**

# Η Υλοποίηση

## Είσοδος – Έξοδος

Η είσοδος και των δύο αλγορίθμων είναι ένα σύνολο σημείων (οποιουδήποτε αριθμού) στον R2. Στην υλοποίησή μας, αυτά τα σημεία επιλέγονται τυχαία με βάση κάποιες παραμέτρους που δίνονται από τον χρήστη, ωστόσο μπορούμε να δώσουμε και όποιες τιμές θέλουμε εμείς για συγκεκριμένα σημεία. Οι παράμετροι που δίνει ο χρήστης είναι, οι διαστάσεις (coord\_range) μέσα στις οποίες πρέπει να βρίσκονται τα σημεία, για παράδειγμα εάν δοθεί ένα εύρος [0, 40] συνεπάγεται ότι τα σημεία θα βρίσκονται στο [0, 40] [0, 40] στον R2, επίσης, ο χρήστης δίνει τον αριθμό των σημείων που θέλει (user\_num\_points).

Τώρα, όσον αφορά τον αλγόριθμο ελαχιστοποίησης εμβαδού ορθογωνίου για δοθέν πλήθος σημείων, ο χρήστης ορίζει το επιθυμητό πλήθος σημείων που θέλει να περικλείονται στο ορθογώνιο (user\_goal\_points). Ενώ, στον αλγόριθμο μεγιστοποίησης πλήθους σημείων για δοθέν εμβαδόν ορθογωνίου, ορίζει το εμβαδόν του ορθογωνίου (user\_area). Παρακάτω δίνεται ο κώδικας για τον πρώτο και δεύτερο αλγόριθμο αντίστοιχα:

coord\_range = [0, 80]

*# the number of points of set P*

user\_num\_points = 180

*# the number of points inside the box, given by the user*

user\_goal\_points = 20

*# dimensions of the block of points*

coord\_range = [0, 40]

*# the number of points to be created*

user\_num\_points = 100

*# the specific area of the box, given by the user*

user\_area = 500

Επιπλέον, δίνεται ο κώδικας που χρησιμοποιείται για την κατασκευή του συνόλου των τυχαίων σημείων:

P = []

*def* create\_set(*num\_points*, *coord\_range*):

    for i in range(0, num\_points):

        new = []

        new.append(random.uniform(coord\_range[0], coord\_range[1]))

        new.append(random.uniform(coord\_range[0], coord\_range[1]))

        P.append(new)

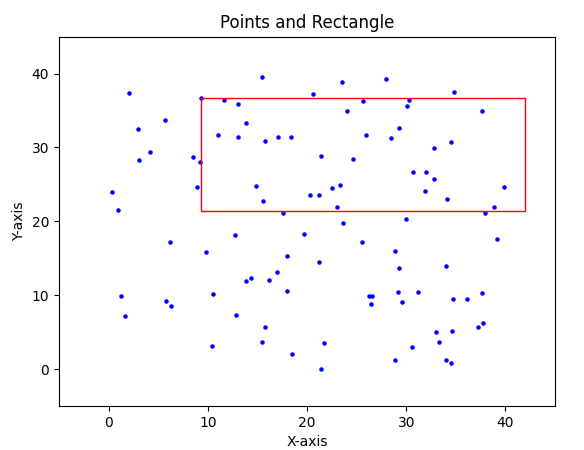
    return P

Η έξοδος και των δύο αλγορίθμων είναι μια γραφική παράσταση των σημείων όλου του συνόλου, που κατασκευάστηκαν τυχαία στην αρχή, μαζί με το ορθογώνιο που ικανοποιεί τις συνθήκες της κάθε περίπτωσης, δηλαδή, για τον πρώτο αλγόριθμο είναι το ορθογώνιο με το ελάχιστο εμβαδόν που περικλείει τα σημεία που έδωσε ο χρήστης. Ενώ για τον δεύτερο είναι πάλι το ορθογώνιο με συγκεκριμένο εμβαδόν που έδωσε ο χρήστης. Ένα παράδειγμα αναπαράστασης της εξόδου φαίνεται στο Σχήμα 3.1. Η αναπαράσταση έχει γίνει με τη βοήθεια της βιβλιοθήκης matplotlib της Python.

## Δομές Δεδομένων

Η αναπαράσταση όλων των σημείων αλλά και των ορθογωνίων έγινε χρησιμοποιώντας πίνακες της Python. Τα σημεία συμβολίζονται ως [[x1, y1], … , [xn, yn]], ενώ το ορθογώνιο ως [xlow, xhigh, ylow, yhigh], όπου xlow, xhigh, ylow και yhigh είναι οι τέσσερις κορυφές του ορθογωνίου.

Για να πετύχουμε την ζητούμενη πολυπλοκότητα του δεύτερου αλγορίθμου χρησιμοποιήσαμε μια ιδιαίτερη μορφή δυαδικού δέντρου, το KD-tree, που μας βοηθά να απαντάμε σε χρόνο O(logn) στο ερώτημα, πόσα σημεία ενός συνόλου P βρίσκονται μέσα σε ένα ορθογώνιο R. Η χρήση αυτής της δομή έγινε με βοήθεια της βιβλιοθήκης spicy της Python.



**Σχήμα 3.1**

## Ελαχιστοποίηση Εμβαδού Ορθογωνίου για Δοθέν Πλήθος Σημείων

Σε αυτήν την ενότητα περιγράφουμε αναλυτικά την κατασκευή του πρώτου αλγορίθμου. Η διαδικασία που ακολουθούμε είναι αυτή που περιγράφεται στην Ενότητα 2.1 της υπάρχουσας εργασίας. Για κάθε λήμμα, υπάρχει και ένα ξεχωριστό αρχείο της γλώσσας Python που επιτελεί τις λειτουργίες του. Επομένως, θα ξεκινήσουμε την ανάλυση από το Λήμμα 1, και θα συνεχίσουμε με τον ίδιο τρόπο.

### Η Συνάρτηση lemma1\_above

Η συνάρτηση αυτή παίρνει ως είσοδο ένα σύνολο σημείων P και για κάθε σημείο p του P επιστρέφει τα k κοντινότερα σημεία στο p με βάση το ορθογώνιο δύο πλευρών μεταξύ του σημείου p και μιας ευθείας l. Εάν δεν υπάρχουν k σημεία που να ικανοποιούν αυτή τη συνθήκη επιστρέφονται k’ < k. Όπως υποδεικνύει και το όνομα της συνάρτησης εξετάζει σημεία που βρίσκονται πάνω από μια ευθεία l και επιστρέφει τα k κοντινότερα αριστερά και τα k κοντινότερα δεξιά ενός σημείου p. Για καλύτερη κατανόηση δείτε και την εξήγηση του Λήμματος 3 της Ενότητας 2.1, που περιγράφει πως ακριβώς χρησιμοποιεί το Λήμμα 1.

*# case == 2: Pq-> where q in (P+ U P-\*)*

*# case == 3: Pq<- where q in (P+ U P-\*)*

*def* lemma1\_above(*setP*, *case*, *num\_points*):

*# points of Rl' set which contains all visited B points*

*# that do not have k points of visitedA*

    unfinished = []

*# coontains the k closest points of each b point (if it has k closest)*

*# because there will be points of b, which will be not have A points to the right of them*

    solution = []

*# which variation of lemma1 i have to run*

    if case == 2:

*# scan space from left to right*

        setP.sort(*key*=*lambda* *x*:x[0])

    elif case == 3:

*# scan space from right to left*

        setP.sort(*key*=*lambda* *x*:x[0], *reverse*=True)

*# sweep-line algorithm*

    for i in setP:

*# its a B set point, so insert it to unfinished array (Rl' set point)*

*# sorted by y-coordinate, so we can use array later to binary\_search*

        if i[2] == 1:

            point = []

            point.append(i[0])

            point.append(i[1])

*# use an empty array to mark the closest points later*

            point.append([])

*# insert the point sorted by its y-coordinate*

            bisect.insort(unfinished, point, *key*=*lambda* *x*:x[1])

*# a point which belongs to A n' B*

        if i[2] == 0:

*# first consider it as a point of B*

            p = []

            p.append(i[0])

            p.append(i[1])

*# use an empty array to mark the closest points later*

            p.append([])

*# insert the point sorted by its y-coordinate*

            bisect.insort(unfinished, p, *key*=*lambda* *x*:x[1])

*# then consider it as a point of A*

            flag = binary\_search(0, len(unfinished)-1, i[1], unfinished)

            if flag != -1:

                toDel = []

                for j in range(flag+1, len(unfinished)):

                    new\_p = []

                    new\_p.append(i[0])

                    new\_p.append(i[1])

                    unfinished[j][2].append(new\_p)

                    if len(unfinished[j][2]) == num\_points:

                        solution.append(unfinished[j])

                        toDel.append(j)

                for d in toDel[::-1]:  *#reversed array toDel*

                    del unfinished[d]

    return solution, unfinished

### Η Συνάρτηση binary\_search

Είναι μια βοηθητική συνάρτηση, παραλλαγή της δυαδικής αναζήτησης, και επιστρέφει δείκτη σε έναν ταξινομημένο πίνακα με βάση την y-συντεταγμένη. Από το δείκτη και μετά θα πρέπει να επεξεργαστούμε κατάλληλα τα στοιχεία του πίνακα με βάση αυτά που εξηγούνται στο Λήμμα 1 της Ενότητας 2.1.

*#returns the place of unfinished array where the point of set A*

*#belongs, accorning to Rl'y < ay*

*#returns -1 if it doesnt belong anywhere*

*def* binary\_search(*low*, *high*, *x*, *unfinished*):

    if high >= low:

        mid = (high+low) // 2

        if high==low:

            if unfinished[high][1] <= x:

                return high

            else:

                return low - 1

        elif unfinished[mid][1] > x:

            return binary\_search(low, mid-1, x, unfinished)

        else:

            return binary\_search(mid+1, high, x, unfinished)

    else:

        if unfinished[low][1] < x:

            return low

        else:

            return high

### Η Συνάρτηση lemma1\_below

Εντελώς αντίστοιχα με τη συνάρτηση της Υπό-Ενότητας 3.3.1, κατασκευάζουμε και την συνάρτηση αυτή, με τη διαφορά ότι ασχολούμαστε με σημεία p που βρίσκονται κάτω από την γραμμή l, όπως υπονοεί και το όνομα της συνάρτησης. Γίνεται χρήση και της συνάρτησης της Υπό-Ενότητας 3.3.2, γι’ αυτό και η εξήγησή της παραλείπεται.

*# case == 0: Pq-> where q in (P- U P+\*)*

*# case == 1: Pq<- where q in (P- U P+\*)*

*def* lemma1\_below(*setP*, *case*, *num\_points*):

*# points of Rl' set which contains all visited B points*

*# that do not have k points of visitedA*

    unfinished = []

*# coontains the k closest points of each b point (if it has k closest)*

*# because there will be points of b, which will be not have A points to the right of them*

    solution = []

*# which variation of lemma1 i have to run*

    if case == 0:

*# scan space from left to right*

        setP.sort(*key*=*lambda* *x*:x[0])

    elif case == 1:

*# scan space from right to left*

        setP.sort(*key*=*lambda* *x*:x[0], *reverse*=True)

*# sweep-line algorithm*

    for i in setP:

*# its a B set point, so insert it to unfinished array (Rl' set point)*

*# sorted by y-coordinate, so we can use array later to binary\_search*

        if i[2] == 1:

            point = []

            point.append(i[0])

            point.append(i[1])

*# use an empty array to mark the closest points later*

            point.append([])

*# insert the point sorted by its y-coordinate*

            bisect.insort(unfinished, point, *key*=*lambda* *x*:x[1])

*# a point which belongs to A n' B*

        if i[2] == 0:

*# first consider it as a point of B*

            p = []

            p.append(i[0])

            p.append(i[1])

*# use an empty array to mark the closest points later*

            p.append([])

*# insert the point sorted by its y-coordinate*

            bisect.insort(unfinished, p, *key*=*lambda* *x*:x[1])

*# then consider it as a point of A*

            flag = binary\_search(0, len(unfinished)-1, i[1], unfinished)

            if flag != -1:

                toDel = []

                for j in range(0, flag+1):

                    new\_p = []

                    new\_p.append(i[0])

                    new\_p.append(i[1])

                    unfinished[j][2].append(new\_p)

                    if len(unfinished[j][2]) == num\_points:

                        solution.append(unfinished[j])

                        toDel.append(j)

                for d in toDel[::-1]:  *#reversed array toDel*

                    del unfinished[d]

    return solution, unfinished

### Η Συνάρτηση lemma2

Η συνάρτηση αυτή επιτελεί την λειτουργία που περιγράφεται στο Λήμμα 2 της Ενότητας 2.1 και υπολογίζει την τιμή της συνάρτησης Φ(Q, q, k) όπου το q είναι σημείο που ανήκει είτε στην κορυφή του ορθογωνίου είτε στη βάση του, ανάλογα με αυτό το σημείο δίνονται και στις επόμενες υπό-ενότητες οι κατάλληλες συναρτήσεις. Αυτό που επιστρέφεται είναι η ελάχιστη τιμή εμβαδού της συνάρτησης Φ() μαζί με τις τέσσερις κορυφές του ορθογωνίου που έχει το συγκεκριμένο εμβαδόν.

*def* lemma2(*array*, *num\_points*):

*#Q\_top sorted by increasing x-coordinate*

    qTopSorted = []

*#Q\_bot sorted by increasing x-coordinate*

    qBotSorted = []

*#the point q*

    q = []

    q.append(array[0][0])

    q.append(array[0][1])

    bestBox = []

*#initially the minimum value is equal to infinity*

    m = *float*('inf')

*# then q is top point*

    if(array[0][1] > array[1][1]):

        qTopSorted = sorted(array, *key*=*lambda* *x*:x[0])

        array = sorted(array, *key*=*lambda* *x*:x[1], *reverse*=True)

        while(len(array) >= num\_points):

*#result is a list of the form of [minimum\_value, solution\_points]*

            result = compute\_best\_top\_box(qTopSorted, q, array[len(array)-1], num\_points)

*#update the total min and the current best points of solution*

            if result[0] < m:

                m = result[0]

                bestBox = result[1].copy()

*#delete the point at the bottom of the set*

            ind = qTopSorted.index(array[len(array)-1])

            qTopSorted.pop(ind)

            array.pop(len(array)-1)

*# q is bottom point of the set*

    else:

        qBotSorted = sorted(array, *key*=*lambda* *x*:x[0])

        array = sorted(array, *key*=*lambda* *x*:x[1])

        while(len(array) >= num\_points):

            result = compute\_best\_bot\_box(qBotSorted, q, array[len(array)-1], num\_points)

            if result[0] < m:

                m = result[0]

                bestBox = result[1].copy()

            ind = qBotSorted.index(array[len(array)-1])

            qBotSorted.pop(ind)

            array.pop(len(array)-1)

    return m, bestBox

### Η Συνάρτηση compute\_best\_top\_box

Δοθέντος ενός συνόλου Qi και το σημείο q που είναι το σημείο με την μεγαλύτερη y-συντεταγμένη στο Qi, όπως περιγράφεται στο Λήμμα 2 της Ενότητας 2.1, επιστρέφει το ορθογώνιο με το ελάχιστο εμβαδόν που προκύπτει με αυτά τα δεδομένα, δηλαδή τις τέσσερις κορυφές του, καθώς και την ελάχιστη αυτή τιμή του εμβαδον.

*#returns the min area box for the current set Qi*

*#and an array with the points of the solution*

*def* compute\_best\_top\_box(*array*, *q\_top*, *q\_bot*, *num\_points*):

*#keep the indexes*

    top\_index = array.index(q\_top)

    bot\_index = array.index(q\_bot)

    best = []

*#initially min box area is infinity*

    minimum = *float*('inf')

*#that means that there is not a box which contains both*

*#top and bottom point*

    if(abs(top\_index - bot\_index) >= num\_points):

        return [minimum, best]

    else:

        for i in range(0,len(array)+1-num\_points):

*#the box it must contains both bottom and top points*

            if((q\_top in array[i:i+num\_points]) and (q\_bot in array[i:i+num\_points])):

                new\_min = (array[i+num\_points-1][0] - array[i][0]) \* (q\_top[1] - q\_bot[1])

*#check if the new area is less than the previous one*

                if(new\_min < minimum):

                    minimum = new\_min

*# store rectangle coordinates in form of [x\_low, x\_high, y\_low, y\_high]*

                    best = [array[i][0], array[i+num\_points-1][0], q\_bot[1], q\_top[1]].copy()

    return [minimum, best]

### Η Συνάρτηση compute\_best\_bot\_box

Κάνει ότι ακριβώς και η συνάρτηση της Υπό-Ενότητας 3.3.5, αλλά αυτή τη φορά εξετάζεται η περίπτωση όπου το σημείο q ανήκει στη βάση του ορθογωνίου που υπολογίζουμε.

*def* compute\_best\_bot\_box(*array*, *q\_bot*, *q\_top*, *num\_points*):

*#keep the indexes*

    top\_index = array.index(q\_top)

    bot\_index = array.index(q\_bot)

    best = []

*#initially min box area is infinity*

    minimum = *float*('inf')

*#that means that there is not a box which contains both*

*#top and bottom point*

    if(abs(top\_index - bot\_index) >= num\_points):

        return [minimum, best]

    else:

        for i in range(0,len(array)+1-num\_points):

*#the box it must contains both bottom and top points*

            if((q\_top in array[i:i+num\_points]) and (q\_bot in array[i:i+num\_points])):

                new\_min = (array[i+num\_points-1][0] - array[i][0]) \* (q\_top[1] - q\_bot[1])

*#check if the new area is less than the previous one*

                if(new\_min < minimum):

                    minimum = new\_min

*#update the points of the best current solution*

                    best = [array[i][0], array[i+num\_points-1][0], q\_bot[1], q\_top[1]].copy()

    return [minimum, best]

### Η Συνάρτηση lemma3

Η συνάρτηση αυτή υπολογίζει και επιστρέφει το σύνολο Qp, όπως αυτό περιγράφεται στο Λήμμα 3 της Ενότητας 2.1, χρησιμοποιώντας τέσσερις φορές τις συναρτήσεις των Υπό-Ενοτήτων 3.3.1 και 3.3.3.

*def* lemma3(*points*, *line*, *num\_points*):

    p\_plus, p\_minus = separate\_set(points, line)

*# below right*

    sol1, un1 = lemma1\_below(p\_minus, 0, num\_points)

    br = sol1 + un1

    br.sort(*key*=*lambda* *x*:x[0])

*# below left*

    sol2, un2 = lemma1\_below(p\_minus, 1, num\_points)

    bl = sol2 + un2

    bl.sort(*key*=*lambda* *x*:x[0])

*# above right*

    sol3, un3 = lemma1\_above(p\_plus, 2, num\_points)

    ar = sol3 + un3

    ar.sort(*key*=*lambda* *x*:x[0])

*# above left*

    sol4, un4 = lemma1\_above(p\_plus, 3, num\_points)

    al = sol4 + un4

    al.sort(*key*=*lambda* *x*:x[0])

    p = sorted(points, *key*=*lambda* *x*:x[0])

    setQ = []

    for i in range(0, len(p)):

        new = []

        new.append(p[i][0])

        new.append(p[i][1])

        v = br[i][2] + bl[i][2] + ar[i][2] + al[i][2]

        new.append(v)

        setQ.append(new)

    return setQ

### Η Συνάρτηση separate\_set

Διαχωρίζει ένα σύνολο P, σε δύο υποσύνολα P+ και P-, με βάση εάν το σημείο p βρίσκεται πάνω ή κάτω από μία ευθεία l, αντίστοιχα. Επιπλέον, το P+ περιέχει και τα συμμετρικά σημεία του P- ως προς τη ευθεία l. Παρόμοια και το P- περιέχει τα συμμετρικά του P+ ως προς την l.

*# separate the points of set P to two sets regarding their place to line l*

*# if above p\_plus , if below p\_min*

*def* separate\_set(*P*, *l*):

*# all points above the line*

    p\_plus = []

*# all points below the line*

    p\_minus = []

    for point in P:

        if point[1] > l:

            p\_minus.append(find\_symmetric(point, l))

*# insert a flag for use in lemma 1*

            point.append(0)

            p\_plus.append(point)

        else:

            p\_plus.append(find\_symmetric(point, l))

*# insert a flag for use in lemma 1*

            point.append(0)

            p\_minus.append(point)

    return p\_plus, p\_minus

### Η Συνάρτηση find\_symmetric

Υπολογίζει και επιστρέφει το συμμετρικό p’, ενός σημείου p ως προς μία ευθεία l.

*# compute the symmetric of a point regarding to line l*

*def* find\_symmetric(*point*, *l*):

    new\_point = []

    new\_point.append(point[0])

    if point[1] > l:

        new\_point.append(l - (point[1] - l))

    else:

        new\_point.append(l + (l - point[1]))

*#insert a flag for use in lemma1*

    new\_point.append(1)

    return new\_point

### 3.3.10 Η Συνάρτηση lemma4

Υπολογίζει και επιστρέφει την τιμή της συνάρτησης V(P, l, k’), όπως περιγράφεται στο Λήμμα 4 της Ενότητας 2.1. Πιο συγκεκριμένα, καλεί την συνάρτηση της Υπό-Ενότητας 3.3.7 για να υπολογίσει τα σύνολα Qp για κάθε στοιχείο του Qp υπολογίζει τη τιμή της συνάρτησης Φ(), του Λήμματος 2 της Ενότητας 2.1 και τέλος επιστρέφει την ελάχιστη τιμή της Φ(), μαζί με τις κορυφές του ορθογωνίου του οποίου το εμβαδόν ισούται με αυτήν την ελάχιστη τιμή. Αυτή η ελάχιστη τιμή της Φ() ταυτίζεται με την τιμή της V().

*def* lemma4(*points*, *line*, *num\_points*):

*# preprocess and creation of the set P*

    setQ = lemma3(points, line, num\_points)

*# area = a big value initially so we can minimize it later*

    area = *float*('inf')

*# best box === box with the minimum area and most points*

    best\_box = []

    for q in setQ:

*# prepare the item of setQ for lemma2*

        p = []

        p.append(q[0])

        p.append(q[1])

        new = []

        new.append(p)

        new = new + q[2]

        if(len(new) >= num\_points):

            area\_new, best\_box\_new = lemma2(new, num\_points)

            if area\_new < area:

                area = area\_new

                best\_box = best\_box\_new.copy()

    return area, best\_box

### 3.3.11 Η Συνάρτηση minimize\_area

Είναι η βασική συνάρτηση του αλγόριθμού μας που επιστρέφει και την βέλτιστη τιμή του εμβαδού. Βασίζεται σε μία υλοποίηση διαίρει-και-βασίλευε, όπου η βάση της αναδρομής είναι η συνθήκη όπου τα σημεία που απομένουν είναι λιγότερα από αυτά που ζητά ο χρήστης να περικλείονται από το ορθογώνιο. Έπειτα, διαιρούμε σε κάθε επανάληψη το σύνολο των σημείων σε δύο υποσύνολα ίδιου μεγέθους με βάση μία γραμμή που υπολογίζεται έτσι ώστε να ισχύει αυτό. Στη συνέχεια, αναδρομικά ψάχνουμε στα δύο υποσύνολα για τυχόν βέλτιστα ορθογώνια. Τέλος, επιστρέφουμε την βέλτιστη τιμή (και τις κορυφές του ορθογωνίου που την ικανοποιούν) που έχει προκύψει από την αναζήτηση και η οποία είτε θα τέμνει την αρχική ευθεία l, είτε θα βρίσκεται σε ένα από τα δύο αρχικά υποσύνολα.

*def* minimize\_area(*points*, *num\_points*):

*# base case scenario*

    if len(points) < num\_points:

*# then we know that there is not an appropriate box*

        return *float*('inf'), []

*# compute median index of sorted points so we can split set*

    med\_index = len(points) // 2

*# line = c*

    line = ((points[med\_index][1] - points[med\_index - 1][1]) / 2) + points[med\_index - 1][1]

*# split set into two sub-sets of roughly equal number of points*

    points\_below = points[:med\_index]

    points\_above = points[med\_index:]

    min\_area\_below, best\_box\_below = minimize\_area(points\_below, num\_points)

    min\_area\_above, best\_box\_above = minimize\_area(points\_above, num\_points)

    min\_current\_area, current\_best = lemma4(points, line, num\_points)

    return find\_min(min\_area\_below, min\_area\_above, min\_current\_area, best\_box\_below, best\_box\_above, current\_best)

### 3.3.12 Η Συνάρτηση find\_min

Είναι μία βοηθητική συνάρτηση, αντί της παρεχόμενης από την γλώσσα προγραμματισμού min(), και την χρησιμοποιούμε για να επιστρέψουμε το ζεύγος (χαμηλότερη\_τιμή\_εμβαδού, κορυφές\_ορθογωνίου).

*def* find\_min(*min\_area\_below*, *min\_area\_above*, *min\_current\_area*, *best\_box\_below*, *best\_box\_above*, *current\_best*):

    if (min\_area\_below <= min\_area\_above) and (min\_area\_below <= min\_current\_area):

        return min\_area\_below, best\_box\_below

    elif (min\_area\_above <= min\_area\_below) and (min\_area\_above <= min\_current\_area):

        return min\_area\_above, best\_box\_above

    else:

        return min\_current\_area, current\_best

### 3.3.13 Η Συνάρτηση plot\_points

Είναι υπεύθυνη για την αναπαράσταση των σημείων και του ορθογωνίου, με χρήση της βιβλιοθήκης matplotlib.

*#rectangle = [x\_low, x\_high, y\_low, y\_high]*

*def* plot\_points(*points*, *rectangle*):

*# create a new figure and axis*

    ax = plt.subplots()

*# define points to be plotted*

    points\_x = [p[0] for p in points]

    points\_y = [p[1] for p in points]

*# plot points*

    ax.scatter(points\_x, points\_y, *s*=5, *color*='blue', *label*='Points')

*# define rectangle parameters (lower-left corner x, y, width, height)*

    rect = patches.Rectangle((rectangle[0], rectangle[2]), rectangle[1] - rectangle[0], rectangle[3] - rectangle[2], *linewidth*=1, *edgecolor*='red', *facecolor*='none')

*# ddd rectangle to the plot*

    ax.add\_patch(rect)

*# set limits for the axes*

    ax.set\_xlim(coord\_range[0] - 5, coord\_range[1] + 5)

    ax.set\_ylim(coord\_range[0] - 5, coord\_range[1] + 5)

*# add labels and title*

    ax.set\_xlabel('X-axis')

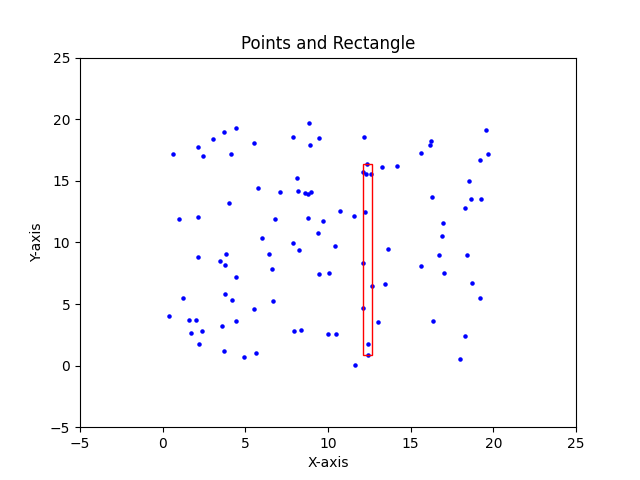
    ax.set\_ylabel('Y-axis')

    ax.set\_title('Points and Rectangle')

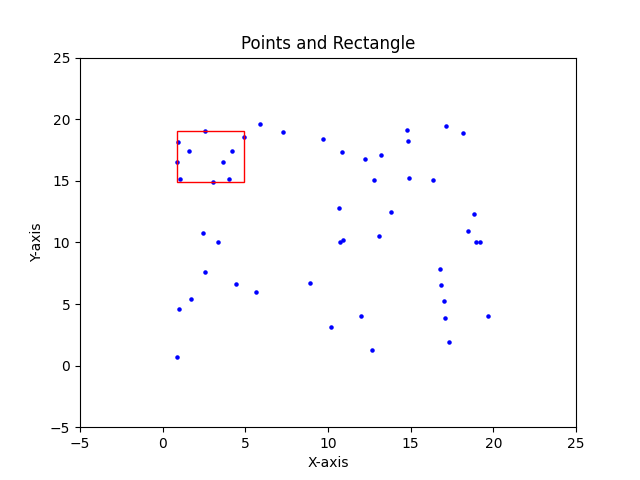
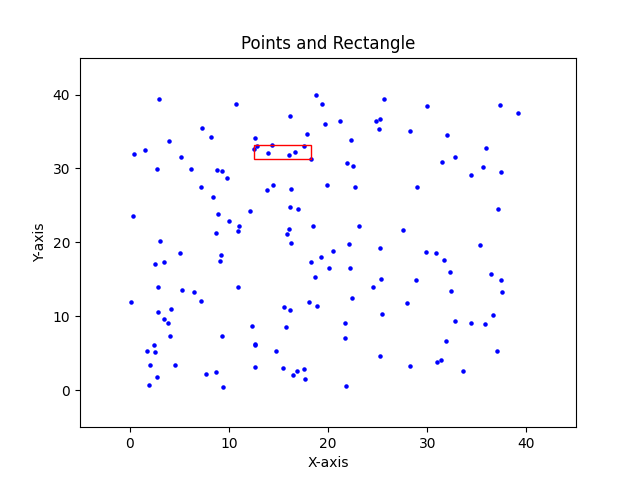
*# display the plot*

    plt.show()

### 3.3.14 Παραδείγματα Εκτέλεσης του Προγράμματος

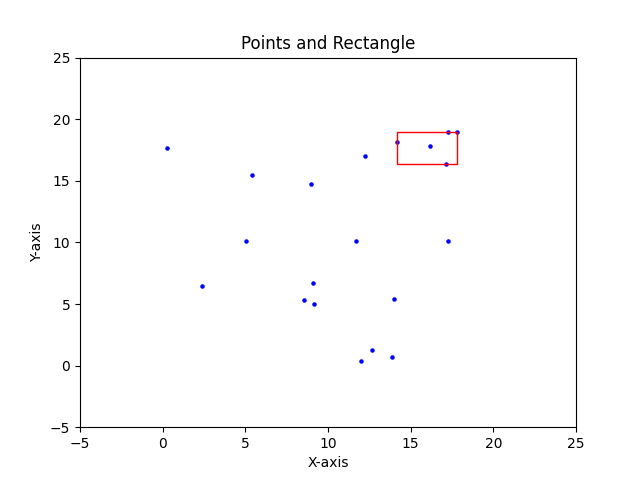
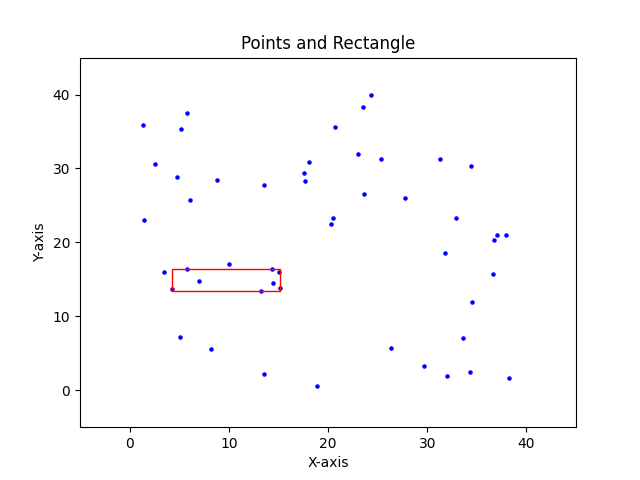
 Στα σχήματα Σχήμα 3.2 έως Σχήμα 3.6 φαίνεται η έξοδος του προγράμματος για διάφορες τιμές εισόδου. Στο Σχήμα 3.7 αναπαρίσταται η έξοδος για το παράδειγμα της Ενότητας 2.1.

**Σχήμα 3.2:** coord\_range=[0, 20], set\_points=100, user\_goal\_points=10 best\_area=8.68



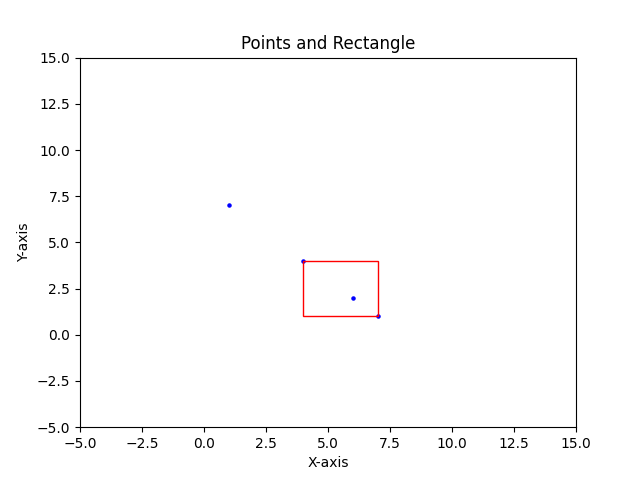
**Σχήμα 3.4:** coord\_range=[0, 40], set\_points=150, user\_goal\_points=8 best\_area=10.96

**Σχήμα 3.3:** coord\_range=[0, 20], set\_points=50, user\_goal\_points=10 best\_area=16.77



**Σχήμα 3.6:** coord\_range=[0, 20], set\_points=20, user\_goal\_points=5 best\_area=9.35

**Σχήμα 3.5:** coord\_range=[0, 40], set\_points=50, user\_goal\_points=8 best\_area=10.96



**Σχήμα 3.7:** Το παράδειγμα της Ενότητας 2.1

## Μεγιστοποίηση του Πλήθους Σημείων για Δοθέν Εμβαδόν Ορθογωνίου

Σε αυτήν την ενότητα περιγράφουμε την υλοποίηση του δεύτερου αλγορίθμου που ακολουθεί την λογική υλοποίησης του πρώτου. Υλοποιούμε έναν 4-προσεγγιστικό αλγόριθμο με χρήση της τεχνικής διαίρει-και-βασίλευε.

### Η Συνάρτηση max\_points\_in\_box

Υπολογίζει και επιστρέφει το μέγιστο αριθμό σημείων που υπάρχουν μέσα σ’ ένα ορθογώνιο συγκεκριμένου εμβαδού α. Γίνεται χρήση ενός αλγορίθμου διαίρει-και-βασίλευε. Η βάση της αναδρομής είναι όταν τα υπολειπόμενα σημεία του συνόλου γίνουν μικρότερα ή ίσα του ένα. Έπειτα, σε κάθε επανάληψη της αναδρομής, χωρίζουμε το σύνολο των σημείων P σε δύο υποσύνολα του ίδιου μεγέθους με βάση μία οριζόντια γραμμή l. Ακολούθως, υπολογίζουμε τη βέλτιστη λύση για το κύριο σύνολο με βάση την γραμμή l και εν συνεχεία, αναδρομικά λύνουμε το πρόβλημα στα δύο υποσύνολα. Στο τέλος, επιστρέφουμε την προσέγγιση του μέγιστου αριθμού σημείων καθώς και το ορθογώνιο εμβαδού α που περικλείει αυτά τα σημεία. Για να κατανοήσουμε καλύτερα την ακρίβεια της προσεγγιστικής λύσης, δίνουμε το παρακάτω παράδειγμα, εάν για ένα δεδομένο εμβαδόν ορθογωνίου ο βέλτιστος αριθμός σημείων που καλύπτονται από αυτό το ορθογώνιο είναι 100, τότε η προσεγγιστική λύση θα έχει τιμή

25 προσεγγιστική\_λύση 100.

*# returns the max number of points inside a box with area a*

*def* max\_points\_in\_box(*points*, *area*):

*# base case of recursion*

    if len(points) <= 1:

        return 1, points

    med\_index = len(points) // 2

*# line = c*

    line = ((points[med\_index][1] - points[med\_index - 1][1]) / 2) + points[med\_index - 1][1]

*# split set into two sub-sets of roughly equal number of points*

    points\_below = points[:med\_index]

    points\_above = points[med\_index:]

    max\_below, box\_below = max\_points\_in\_box(points\_below, area)

    max\_above, box\_above = max\_points\_in\_box(points\_above, area)

    max\_current, box\_current = current\_max\_inside(line, points, area)

    return find\_max(max\_below, max\_above, max\_current, box\_below, box\_above, box\_current)

### Η Συνάρτηση current\_max\_inside

Επιστρέφει το ορθογώνιο με εμβαδό α, από ένα σύνολο ορθογωνίων, που περιέχει τα περισσότερα σημεία ενός συνόλου P, καθώς και το πλήθος των σημείων αυτών. Το σύνολο ορθογωνίων είναι αυτό που περιγράφεται στην Ενότητα 2.2 ως Rα(l). Αφού είναι γνωστές οι συντεταγμένες κάποιου σημείου p P, δεδομένου του σταθερού εμβαδού α αλλά και της ευθείας l μπορούμε να υπολογίσουμε εύκολα τις κορυφές του ορθογωνίου που προκύπτει.

*# returns the maximum number of points inside a box*

*# considering the curremt line and the four boxes explained in paper*

*def* current\_max\_inside(*line*, *points*, *area*):

    count = 0

    best\_box = []

    for p in points:

*# point above line l*

        if p[1] > line:

*# we know that area = (y\_high - y\_low)\*(x\_high - x\_low), so we compute the needed values*

            x\_low = p[0]

            x\_high = (area + (p[0] \* (p[1] - line))) / (p[1] - line)

            y\_low = line

            y\_high = p[1]

            count\_right = count\_points\_inside(x\_low, x\_high, y\_low, y\_high)

            if count\_right > count:

                count = count\_right

                best\_box = [x\_low, x\_high, y\_low, y\_high].copy()

            x\_low = ((p[0] \* (p[1] - line)) - area) / (p[1] - line)

            x\_high = p[0]

            count\_left = count\_points\_inside(x\_low, x\_high, y\_low, y\_high)

            if count\_left > count:

                count = count\_left

                best\_box = [x\_low, x\_high, y\_low, y\_high].copy()

        if p[1] < line:

            x\_low = p[0]

            x\_high = (area + (p[0] \* (line - p[1]))) / (line - p[1])

            y\_low = p[1]

            y\_high = line

            count\_right = count\_points\_inside(x\_low, x\_high, y\_low, y\_high)

            if count\_right > count:

                count = count\_right

                best\_box = [x\_low, x\_high, y\_low, y\_high].copy()

            x\_low = ((p[0] \* (line - p[1])) - area) / (line - p[1])

            x\_high = p[0]

            count\_left = count\_points\_inside(x\_low, x\_high, y\_low, y\_high)

            if count\_left > count:

                count = count\_left

                best\_box = [x\_low, x\_high, y\_low, y\_high].copy()

    return count, best\_box

### Η Συνάρτηση find\_max

Επιστρέφει το ζεύγος (μέγιστο\_πλήθος\_σημείων, ορθογώνιο).

*def* find\_max(*max\_below*, *max\_above*, *max\_current*, *box\_below*, *box\_above*, *box\_current*):

    if (max\_below >= max\_above) and (max\_below >= max\_current):

        return max\_below, box\_below

    elif (max\_above >= max\_below) and (max\_above >= max\_current):

        return max\_above, box\_above

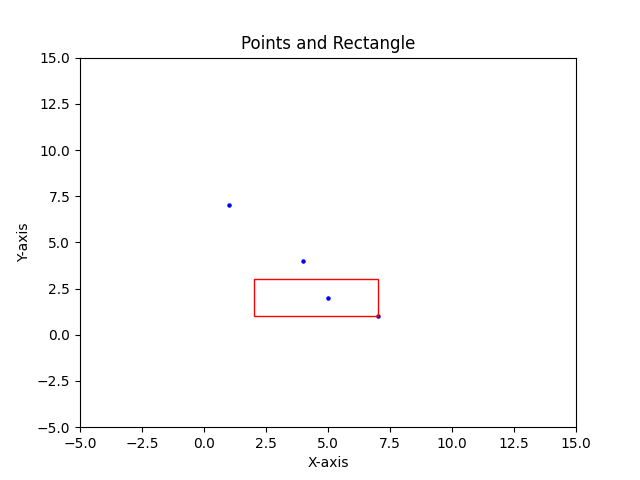
    else:

        return max\_current, box\_current

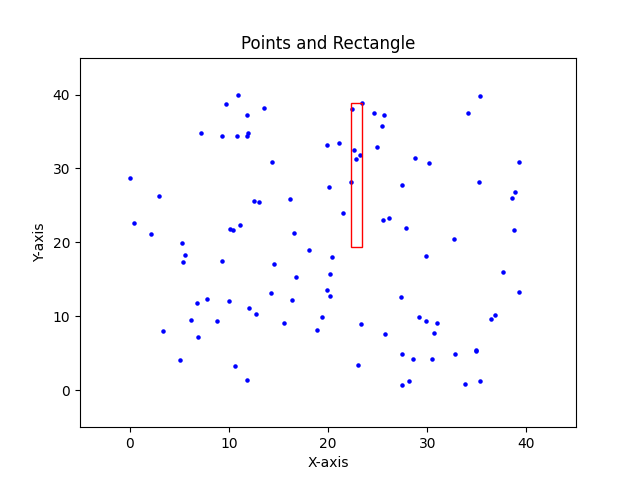
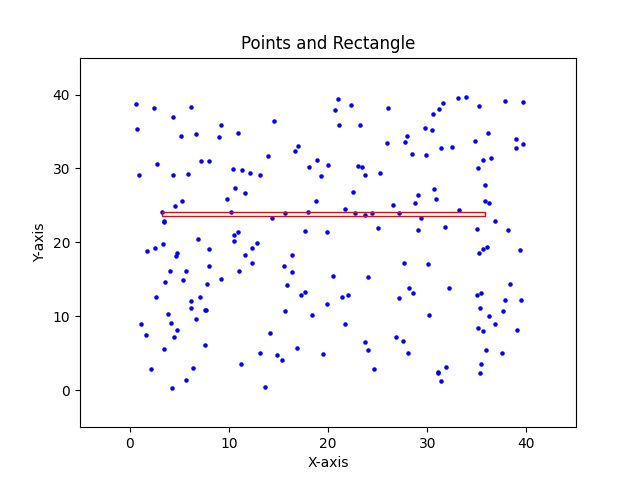
### Η Συνάρτηση plot\_points

Είναι υπεύθυνη για την αναπαράσταση των σημείων και του ορθογωνίου κάνοντας χρήση της βιβλιοθήκης matplotlib της Python. Ο κώδικας παραλείπεται γιατί είναι ίδιος με αυτόν της Υπό-Ενότητας 3.3.13.

### Παραδείγματα Εκτέλεσης του Προγράμματος

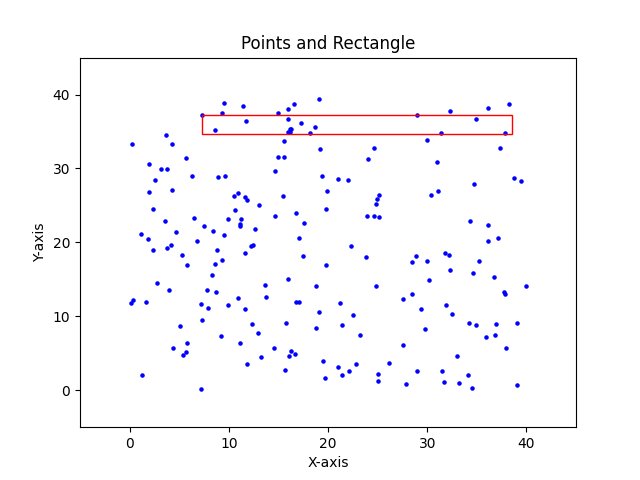
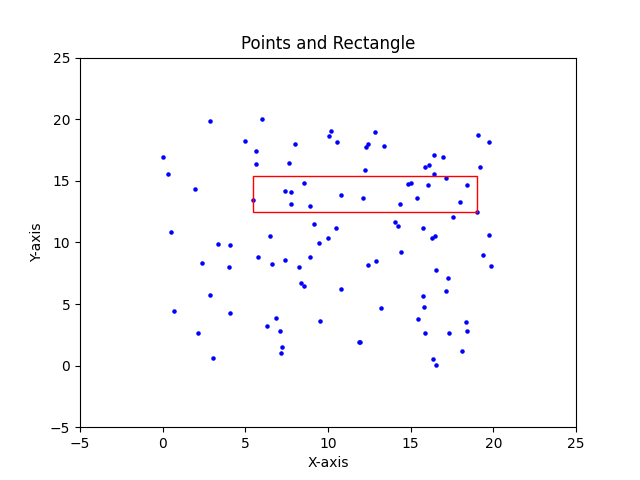
Στα σχήματα Σχήμα 3.9 έως Σχήμα 3.12, φαίνεται η έξοδος του προγράμματος για διάφορες τιμές εισόδου. Στο Σχήμα 3.8 αναπαρίσταται η έξοδος για το παράδειγμα που περιεγράφηκε στην Ενότητα 2.2.

**Σχήμα 3.8:** Το παράδειγμα της Ενότητας 2.2



**Σχήμα 3.8:** coord\_range=[0, 40], set\_points=200, user\_area=20 number\_of\_points=8

**Σχήμα 3.7:** coord\_range=[0, 40], set\_points=100, user\_area=20 number\_of\_points=6



**Σχήμα 3.10:** coord\_range=[0, 20], set\_points=100, user\_area=40 number\_of\_points=17

**Σχήμα 3.9:** coord\_range=[0, 40], set\_points=200, user\_area=80 number\_of\_points=14

# Επίλογος

Σε αυτή τη διπλωματική εργασία πραγματευθήκαμε το πρόβλημα της κάλυψης σημείων με ένα ορθογώνιο μικρού εμβαδού. Πιο συγκεκριμένα, μελετήσαμε και υλοποιήσαμε στη γλώσσα προγραμματισμού Python τους αλγόριθμους των Berg, Cabello, Cheong, Eppstein και Knauer για το παραπάνω πρόβλημα [6]. Κατασκευάσαμε έναν αλγόριθμο χρόνου O(nk2logn + nlog2n) για το πρόβλημα ελαχιστοποίησης εμβαδού ορθογωνίου για δοθέν πλήθος σημείων, ενώ παρουσιάσαμε ένα αλγόριθμο χρόνου O(nlog2n) για το πρόβλημα μεγιστοποίησης του πλήθους σημείων για δοθέν εμβαδόν ορθογωνίου. Σίγουρα υπάρχουν περιθώρια βελτίωσης για τους αλγορίθμους και μπορεί να βρεθεί κάποιος ταχύτερος βελτιώνοντας κάποιο από τα λήμματα που παρουσιάσαμε στο Κεφάλαιο 2. Παρόλα αυτά, αυτή τη στιγμή, το πρόβλημα επινόησης ταχύτερου αλγορίθμου για το πρόβλημα του πρώτου αλγορίθμου παραμένει ανοιχτό προς έρευνα.

Βιβλιογραφία

|  |  |
| --- | --- |
| [1] | Alok Aggarwal, Hiroshi Imai, Naoki and Subhash Suri. Finding k-points with minimum diameter and related problems. J. Algorithms, 12(1):38-56, 1991. |
| [2] | Hee-Hap Ahn, Sang Won Bae, Erik D. Demaine, Martin L. Demaine, Sang-Sub Kim, Matias Korman, Iris Reinbacher and Wanbin Son. Covering points by disjoint boxes with outliers. Comput. Geom., 44(3):178-190, 2011.. |
| [3] | Boris Aronov, Esther Ezra and Micha Sharir. Small-size ε-nets for axis-parallel rectangles and boxes. SIAM J. Comput., 39(7):3248-3282, 2010. |
| [4] | Sandip Das, Partha P. Goswami and Subhas C. Nandy. Smallest k-point enclosing rectangle and square of arbitrary orientation. Inf. Process. Lett., 94(6):259-266, 2005 |
| [5] | Amitava Datta, Hans-Peter Lenhof, Christian Schwarz and Michiel H. Smid. Static and dynamic algorithms for k-point clustering problems. J. Algorithms, 19(3):474-503, 1995. |
| [6] | Mark de Berg, Sergio Cabello, Otfried Cheong, David Eppstein and Christian Knauer. Covering many points with a small-area box. JoCG 10(1), 207-222, 2019.. |
| [7] | David Eppstein and Jeff Erickson. Iterated nearest neighbors and finding minimal polytopes. Discrete Comput. Geom., 11(3):321-350, 1994. |
| [8] | Haim Kaplan, Sasanka Roy and Micha Sharir. Finding axis-parallel rectangles of fixed perimeter or area containing the largest number of points. In Proc. 25th Annual European Symposium (ESA), volume 87 of LIPIcs, pages 52:1-52:13. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2017. |
| [9] | Michael Segal and Klara Kedem. Enclosing k points in the smallest axis parallel rectangle. Inform. Process. Lett., 65(2):95-99, 1998 |
| [10] | Dan E. Willard. New data structures for orthogonal range queries. SIAM J. Comput., 14(1):232-253, 1985. |