Υλοποίηση Αλγορίθμων για την Κάλυψη Σημείων με ένα Ορθογώνιο Μικρού Εμβαδού

Σπυρίδων Σιλίρας

Διπλωματική Εργασία

Επιβλέπων Καθηγητής: Λεωνίδας Παληός

Ιωάννινα, Σεπτέμβριος, 2024

ΤΜΗΜΑ ΜΗΧ. Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

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 $O(n\ k^2 \log n + n \log^2 n)$. Επίσης, θεωρούμε το πρόβλημα για το οποίο δοθέντος μιας τιμής $\alpha > 0$, 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 $O(n \log^2 n)$.

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 $O(n k^2 \log n + n \log^2 n)$ time. We also consider the problem of, given a value $\alpha > 0$, 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 $O(n \log^2 n)$ time.

Keywords: Rectangle, Area, 4-approximation

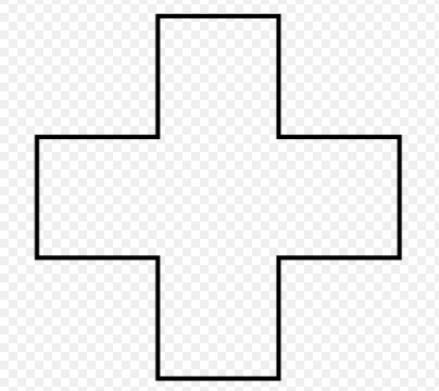
Πίνακας περιεχομένων

Κεφάλαιο 1. Εισαγωγή

1.1 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 *ορθογώνιου* (orthogonal) *πολυγώνου*, του *κλιμακωτού* (staircase) *πολυγώνου*, της *ορατότητας* (visibility), της *k-ορατότητας* (k-visibility), του *1-φύλακα* (1-modem), του *k-φύλακα* (k-modem) καθώς και η έννοια της *φύλαξης πολυγώνου από k-φύλακες* για k >= 1.

Το *ορθογώνιο* (orthogonal) *πολύγωνο* είναι ένα πολύγωνο του οποίου οι ακμές είναι παράλληλες προς τους άξονες του καρτεσιανού συστήματος συντεταγμένων (Σχήμα 1.1).



Σχήμα 1.1: ορθογώνιο πολύγωνο

Το *κλιμακωτό* (staircase) *πολύγωνο* είναι μια περιορισμένη εκδοχή του ορθογώνιου πολυγώνου, όπου έχει τη μορφή ‘’σκάλας’’, όλες οι εσωτερικές του γωνίες είναι 90° και 270°, είναι xy-μονότονο και οι ‘’σκάλες’’ επιτρέπεται να είναι μόνο ανοδικές και προς τα δεξιά. Για ευκολία παρουσιάζεται το παρακάτω σχήμα (Σχήμα 1.2).

Σχήμα 1.2: κλιμακωτό πολύγωνο

Με τον όρο *ορατότητα* (visibility) αναφερόμαστε στο γεγονός όπου δύο σημεία μπορούν να συνδεθούν με μία ευθεία γραμμή χωρίς να παρεμβάλλεται κάποιο εμπόδιο μεταξύ τους. Επιπλέον, λέμε ότι δύο σημεία έχουν *k-ορατότητα* (k-visibility) μεταξύ τους, όταν παρεμβάλλονται τουλάχιστον k εμπόδια μεταξύ της νοητής ευθείας που τα ενώνει.

Με την έννοια του *1-φύλακα* (1-modem) αναφερόμαστε σ' έναν μοναδικό φύλακα που τοποθετείται σε κάποιο σημείο ενός πολυγώνου και έχει την ικανότητα να το επιτηρεί πλήρως, δηλαδή η γραμμή οπτικής επαφής από τη θέση του φύλακα προς κάθε σημείο του πολυγώνου δεν πρέπει να τέμνει τα εσωτερικά τοιχώματα του πολυγώνου.

Η έννοια του *k-φύλακα* (k-modem) αναφέρεται σε μια ομάδα από k φύλακες που τοποθετούνται σε στρατηγικά σημεία εντός ενός πολυγώνου με σκοπό να επιτηρούν πλήρως το εσωτερικό του. Κάθε φύλακας έχει οπτική επαφή με το μέρος του πολυγώνου που είναι ευθύγραμμα ορατό από το σημείο που βρίσκεται, χωρίς εμπόδια από τις πλευρές του πολυγώνου. Οι φύλακες πρέπει να τοποθετηθούν με τέτοιον τρόπο ώστε οι περιοχές που βλέπουν συλλογικά να καλύπτουν πλήρως το εσωτερικό του πολυγώνου.

Όταν αναφερόμαστε στην έννοια *φύλαξης πολυγώνου από k-φύλακες*, εννοούμε το πρόβλημα υπολογισμού ενός ελάχιστου αριθμού φύλακων που χρειάζεται για να επιτηρούν πλήρως ένα πολύγωνο. Το πολύγωνο μπορεί να έχει οποιοδήποτε σχήμα, ακόμη και μη κυρτό. Οι φρουροί τοποθετούνται σε σημεία εντός του πολυγώνου και μπορούν να βλέπουν οποιοδήποτε σημείο είναι ευθύγραμμα ορατό από τη θέση τους, δηλαδή αν η γραμμή μεταξύ του φρουρού και του σημείου δεν τέμνει τα όρια του πολυγώνου.

1.2 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.

1.3 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 $O(k^2 n \log n)$ και $O(n \log n + k^2 n)$ 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 $O(n^{5/2} \log^2 n)$. 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 $O(nk^2 \log n + n \log^2 n)$ 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 $O(n \log^2 n)$ αλγόριθμο, που χρησιμοποιεί με παρόμοιο τρόπο την τεχνική διαίρει-και-βασίλευε.

1.4 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.

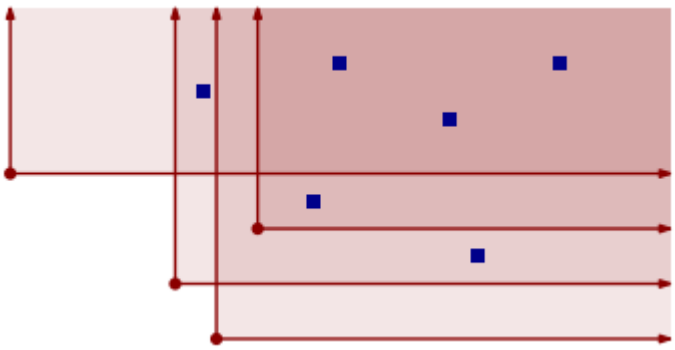
Κεφάλαιο 2. Αλγόριθμοι Κάλυψης Σημείων

2.1 Ελαχιστοποίηση Εμβαδού Ορθογωνίου για Δοθέν Πλήθος Σημείων

Αρχικά, θεωρούμε ένα σύνολο P των n σημείων στο χώρο R^2 καθώς και έναν ακέραιο $k \geq 2$, 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 $p \in R^2$, χρησιμοποιούμε τα σύμβολα p_x και p_y για τις x- και y-συντεταγμένες, αντίστοιχα. Ακόμη, το ελάχιστο εμβαδό που ζητείται συμβολίζεται ως

$$\text{area}^*(P, k) = \min \{ \text{area}(R) \mid R \text{ is a box with } |R \cap P| \geq k \}.$$

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.

Λήμμα 1. *Θεωρούμε δύο σύνολα A και B με τουλάχιστον n σημεία στο χώρο R^2. Για κάθε σημείο $b \in B$, ορίζουμε ως R_b το ορθογώνιο δύο πλευρών $[b_x, \infty) \times [b_y, \infty)$, όπως φαίνεται και στο παραπάνω σχήμα (Σχήμα 2.1). Σε χρόνο O(kn + nlogn) μπορούμε να βρούμε, για όλα τα $b \in B$, τα k σημεία στο $A \cap R_b$ 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 A_l και B_l τα σημεία που βρίσκονται αριστερά της ευθείας l και ανήκουν στα σύνολα A και B, αντίστοιχα. Θεωρούμε την οικογένεια ορθογωνίων $R_l = \{ R_b \mid b \in B_l \}$. Ανά πάσα στιγμή, διατηρούμε το υποσύνολο R_l' του συνόλου R_l, που αποτελείται από ορθογώνια που δεν περιέχουν k σημεία του A_l. Τα ορθογώνια $R_b \in R_l'$ αποθηκεύονται σε ένα ισορροπημένο δέντρο δυαδικής αναζήτησης T, το οποίο είναι ταξινομημένο με βάση την τιμή b_y. Επιπλέον, για κάθε ορθογώνιο $R_b \in R_l'$, αποθηκεύουμε σε μία λίστα L_b τα σημεία του A_l που περιέχει καθώς επίσης και το μέγεθος της λίστας L_b, που το παίρνουμε υπολογίζοντας την τιμή $|R_b \cap A_l|$.

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

Όταν η γραμμή συναντήσει σημείο $b \in B$, τότε το R_b γίνεται μέλος του συνόλου R_l 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 R_b.

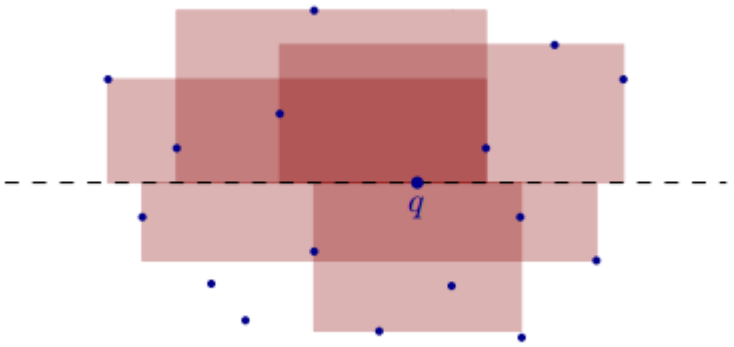
Κάθε εισαγωγή ή διαγραφή στο T παίρνει O(logn). Κάνουμε |B| εισαγωγές και τουλάχιστον |B| διαγραφές στο T, σε συνολικό χρόνο O(nlogn). Για κάθε σημείο $\alpha \in A$, ξοδεύουμε χρόνο O(logn) συν O(1) για κάθε ορθογώνιο R_b το οποίο περιέχει το α. Ο συνολικός χρόνος εκτέλεσης είναι O(kn + nlogn).

Ο παρακάτω ορισμός θα μας χρειαστεί στο Λήμμα 2:

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

$\Phi(Q, q, k) := \min \{$ εμβαδόν(R) $\mid$ όπου το R είναι ένα ορθογώνιο με το $q \in \text{top}(R)$ ή $q \in \text{bot}(R)$ 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.



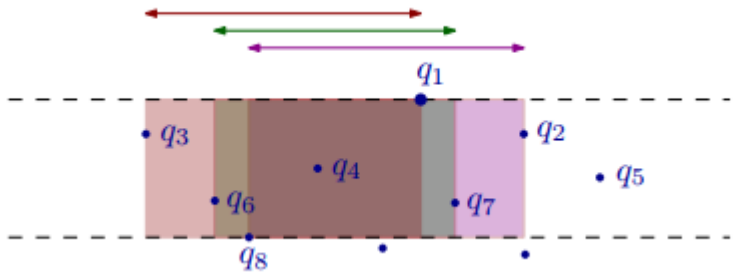
Σχήμα 2.2: διάφορα ορθογώνια που υπολογίζονται κατά τον υπολογισμό της συνάρτησης Φ(Q, q, k) για k=5.

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 q_1, q_2,..., q_m είναι τα σημεία του Q για τα οποία η y-συντεταγμένη δεν είναι μεγαλύτερη από q_y, σε φθίνουσα σειρά με βάση την y-συντεταγμένη. Επίσης, θεωρούμε $Q_i = \{q_1,..., q_i\}$.

Ταξινομούμε τα στοιχεία του Q_i σε αύξουσα σειρά με βάση την x-συντεταγμένη, τότε μπορούμε να βρούμε σε χρόνο $O(|Q_i|) = O(i)$ το ορθογώνιο R με το ελάχιστο εμβαδόν που περιέχει k σημεία, δεδομένου ότι q ∈ top(R) και q_i ∈ 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 Q_m 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 Q_i (αρχικά i = m) σε χρόνο O(i), μετά αφαιρούμε από την λίστα το στοιχείο με την μικρότερη y-συντεταγμένη για να πάρουμε το σύνολο Q_{i-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 Q_8 = $\{q_1,..., q_8\}$.

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

$\Psi(P, l, k) := \min \{$ 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 $\text{area}^*(P, k) \leq \Psi(P, l, k)$. 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).

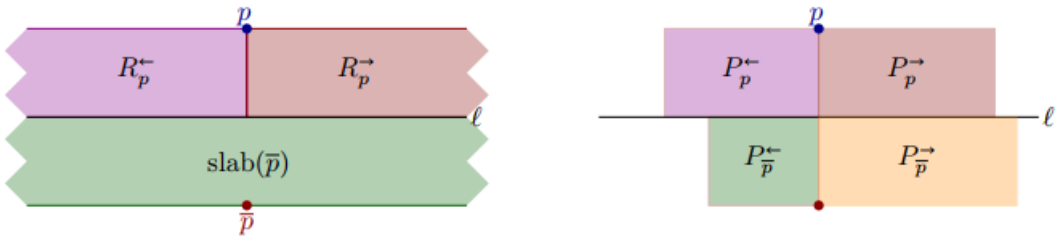
Λήμμα 3. *Δοθέντος των P, l και k, μπορούμε να υπολογίσουμε σε χρόνο O(kn + nlogn) υποσύνολα του P, Q_p, που δεικτοδοτούνται από το $p \in P$, με τις εξής ιδιότητες:*

- *Το Q_p έχει O(k) σημεία για κάθε $p \in P$.*
- *Για κάθε $k' \leq k$, εάν $\text{area}^*(P, k) = \Psi(P, l, k)$, τότε $\text{area}^*(P, k) = \Phi(Q_p, p, k')$ για κάποιο $p \in P$.*

Απόδειξη. Για κάθε p ∈ P, ορίζουμε ως p' το συμμετρικό του p ως προς την ευθεία l. Για κάθε σημείο q του χώρου, όπου $q \notin l$, 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 $R_q^{\rightarrow}$ το ορθογώνιο τριών πλευρών slab(q) ∩ $\{ (x,y) \in R^2 \mid x \geq q_x \}$ και επιπλέον ορίζουμε ως $P_q^{\rightarrow}$ τα k σημεία του P με την μικρότερη χ-συντεταγμένη μέσα στο $R_q^{\rightarrow}$.
- Ορίζουμε ως $R_q^{\leftarrow}$ το ορθογώνιο τριών πλευρών slab(q) ∩ $\{ (x,y) \in R^2 \mid x \leq q_x \}$ και επιπλέον ορίζουμε ως $P_q^{\leftarrow}$ τα k σημεία του P με την μεγαλύτερη χ-συντεταγμένη μέσα στο $R_q^{\leftarrow}$.

Για καλύτερη κατανόηση αυτών τον όρων δείχνουμε το Σχήμα 2.4.



Σχήμα 2.4: η ορολογία που χρησιμοποιούμε στο Λήμμα 3.

Για κάθε p ∈ P, ορίζουμε ως Q_p την ένωση των $P_p^{\rightarrow}$, $P_p^{\leftarrow}$, $P_{p'}^{\rightarrow}$ και $P_{p'}^{\leftarrow}$. Είναι ξεκάθαρο ότι κάθε σετ Q_p 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 Q_{t^*}. Θα το αποδείξουμε με απαγωγή σε άτοπο. Έστω ότι το R* ∩ P περιέχει σημείο α που δεν ανήκει στο Q_{t^*}. 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 Q_{t^*}, δηλαδή σε κάποιο από τα $R_{t^*}^{\rightarrow}$, $R_{t^*}^{\leftarrow}$, $R_{t^{*'}}^{\rightarrow}$ και $R_{t^{*'}}^{\leftarrow}$. Έστω ότι R'' ∈ { $R_{t^*}^{\rightarrow}$, $R_{t^*}^{\leftarrow}$, $R_{t^{*'}}^{\rightarrow}$, $R_{t^{*'}}^{\leftarrow}$ } είναι το ορθογώνιο τριών πλευρών που περιέχει το σημείο α, P'' ∈ { $P_{t^*}^{\rightarrow}$, $P_{t^*}^{\leftarrow}$, $P_{t^{*'}}^{\rightarrow}$, $P_{t^{*'}}^{\leftarrow}$ } 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

$$|t_x^* - q_x''| < |t_x^* - \alpha_x|.$$

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

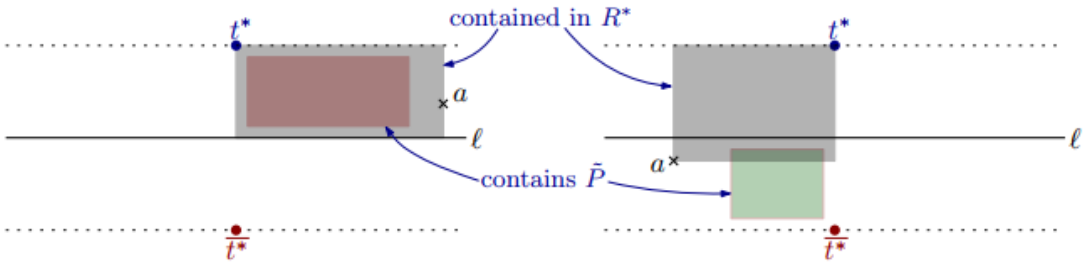
$$|t_x^* - q_x''| \cdot \text{dist}(t^*, l)$$

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

$$\text{area}(R^*) \geq |t_x^* - \alpha_x| \cdot \text{dist}(t^*, l) > |t_x^* - q_x''| \cdot \text{dist}(t^*, l) \geq \text{area}(\text{bb}(P'')).$$

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 Q_{t^*}, και επομένως ισχύει και η δεύτερη ιδιότητα του λήμματος.

Τέλος, απομένει να αποδείξουμε ότι η κατασκευή των συνόλων Q_p, 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 $P_q^{\rightarrow}$ για κάθε q ∈ $P^- \cup P^{+'}$, υπολογίζονται χρησιμοποιώντας το Λήμμα 1 για A = P^- και B = $P^- \cup P^{+'}$. Τα σύνολα $P_q^{\rightarrow}$ για q ∈ $P^+ \cup P^{-'}$, $P_q^{\leftarrow}$ για q ∈ $P^- \cup P^{+'}$ και $P_q^{\leftarrow}$ για q ∈ $P^+ \cup P^{-'}$, υπολογίζονται με παρόμοιο τρόπο χρησιμοποιώντας συμμετρικές εκδοχές του Λήμματος 1.



Σχήμα 2.5: μέρος της απόδειξης του Λήμματος 3.

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 Q_p, 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 $\Phi(Q_p, p, k')$ σε χρόνο $O(|Q_p|^2) = O(k^2)$. Έπειτα, επιστρέφουμε την τιμή $V(P, l, k') = \min \{ \Phi(Q_p, p, k') \mid p \in P \}$. Ο υπολογισμός παίρνει χρόνο $O(nk^2)$.

Αφού για κάθε p ∈ P η τιμή $\Phi(Q_p, p, k')$ 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 $\text{area}^*(P, k') = \Phi(Q_{p0}, p_0, k')$ για κάποιο $p_0 \in P$ και συνεπώς

$$\text{area}^*(P, k') = \Phi(Q_{p0}, p_0, k') \geq \min \{ \Phi(Q_p, p, k') \mid p \in P \} = V(P, l, k').$$

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

$$\text{area}^*(P, k') = \min \{ \Psi(P, l, k'), \text{area}^*(P^+, k'), \text{area}^*(P^-, k') \}$$

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 $O(nk\log n + n\log^2 n)$ 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 $O(nk^2)$, χρησιμοποιώντας το Λήμμα 4. Στη συνέχεια, αναδρομικά υπολογίζουμε τις τιμές area*(P^+, k’) και area*(P^-, k’). Τελικά, επιστρέφουμε την τιμή

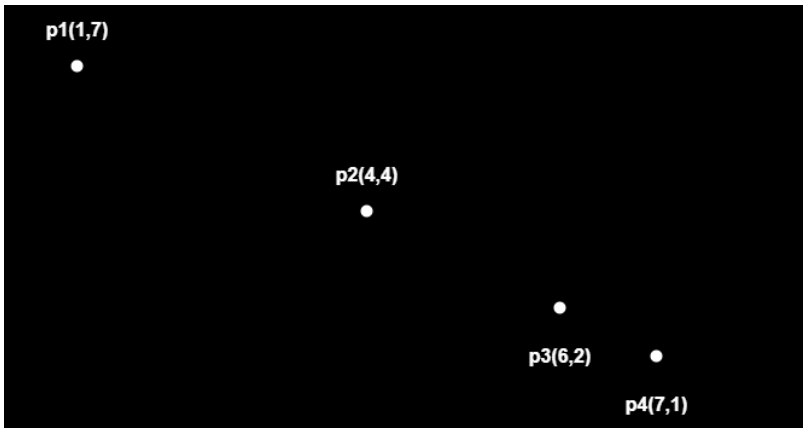
$$\text{area*}(P, k') = \min \{ V(P, l, k'), \text{area*}(P^+, k'), \text{area*}(P^-, k') \}.$$

Γνωρίζοντας ότι έχουμε logn αναδρομικά επίπεδα, ο αλγοριθμός μας τρέχει σε χρόνο $O(nk^2\log n)$.

Συμπέρασμα 6. *Δοθέντος ενός συνόλου n σημείων P και μιας τιμής k μπορούμε να βρούμε σε χρόνο $O(nk^2\log n + n\log^2 n)$ 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 $\Phi(Q_p, p, k)$ αμέσως μετά τον υπολογισμό των συνόλων Q_p.

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.

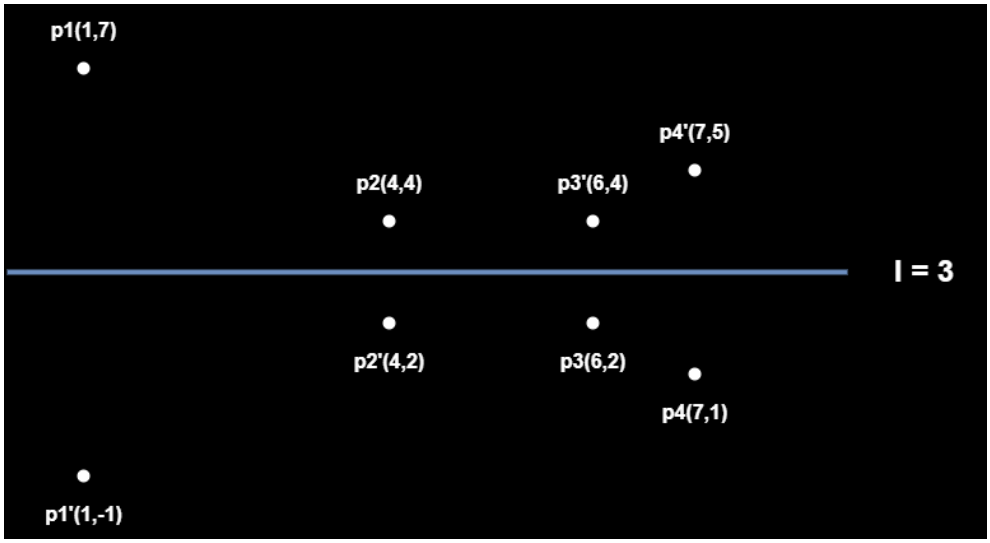


Σχήμα 2.6

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

$$l = (y_{p2} + y_{p3}) / 2 = 3$$

Στο Σχήμα 2.7 φαίνονται και τα συμμετρικά των σημείων ως προς την ευθεία l. Τα υπολογίζουμε γιατί τα χρειαζόμαστε στην κατασκευή των συνόλων Q_p.



Σχήμα 2.7

Έπειτα, υπολογίζουμε την τιμή $V(P, l=3, k=3) = \min \{ \Phi(Q_p, p, k') \mid p \in P \}$. Επομένως, υπολογίζουμε τα σύνολα Q_p ως εξής:

$P_{p1}^{\rightarrow} = [p_1, p_2]$, $P_{p1}^{\leftarrow} = [p_1]$, $P_{p1'}^{\rightarrow} = [p_3, p_4]$, $P_{p1'}^{\leftarrow} = []$. Άρα, $Q_{p1} = [p_1, p_2, p_3, p_4]$.

$P_{p2}^{\rightarrow} = [p_2]$, $P_{p2}^{\leftarrow} = [p_2]$, $P_{p2'}^{\rightarrow} = [p_3]$, $P_{p2'}^{\leftarrow} = []$. Άρα, $Q_{p2} = [p_2, p_3]$.

$P_{p3}^{\rightarrow} = [p_3]$, $P_{p3}^{\leftarrow} = [p_3]$, $P_{p3'}^{\rightarrow} = []$, $P_{p3'}^{\leftarrow} = [p_2]$. Άρα, $Q_{p2} = [p_2, p_3]$.

$P_{p4}^{\rightarrow} = [p_4]$, $P_{p4}^{\leftarrow} = [p_4, p_3]$, $P_{p4'}^{\rightarrow} = []$, $P_{p4'}^{\leftarrow} = [p_2]$. Άρα, $Q_{p2} = [p_2, p_3, p_4]$.

Τα σύνολα Q_{p2} και Q_{p3}, δεν μπορούν να μας δώσουν κάποια πιθανή λύση γιατί περιέχουν δύο σημεία που είναι λιγότερα του k = 3.

Στη συνέχεια υπολογίζουμε τις τιμές $\Phi(Q_p, p, k)$ για τα σύνολα Q_{p1} και Q_{p4}:

- $\Phi(Q_{p1}, p_1, k=3) = (x_{p3} - x_{p1}) \cdot (y_{p1} - y_{p3}) = 5 \cdot 5 = 25$
  Το εμβαδόν το παίρνουμε αφού έχουμε αφαιρέσει από το Q_{p1} το p_4, που έχει την μικρότερη y-συντεταγμένη, για να έχουμε ορθογώνιο που καλύπτει τρία σημεία.
- $\Phi(Q_{p4}, p_4, k=3) = (x_{p4} - x_{p2}) \cdot (y_{p2} - y_{p1}) = 3 \cdot 3 = 9$.

Τελικά, έχουμε V(P, l=3, k=3) = min { $\Phi(Q_{p1}, p_1, k=3)$, $\Phi(Q_{p4}, p_4, k=3)$ } = 9 και το ορθογώνιο που προκύπτει ορίζεται ως

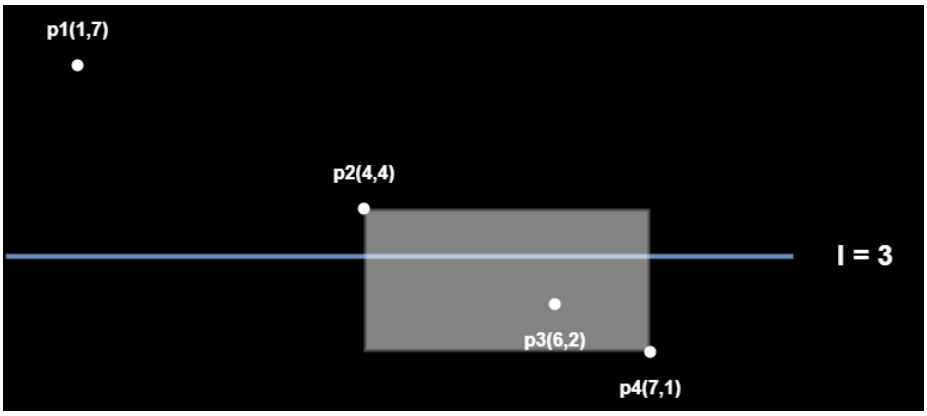
$[x_{low}, x_{high}, y_{low}, y_{high}] = [4, 7, 1, 4]$.

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

$$\begin{aligned} area^*(P, k) &= \min \{ V(P, l, k), area^*(P^+, k), area^*(P^-, k) \} \\ &= \min \{ 9, +\infty, +\infty \} \\ &= 9. \end{aligned}$$

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



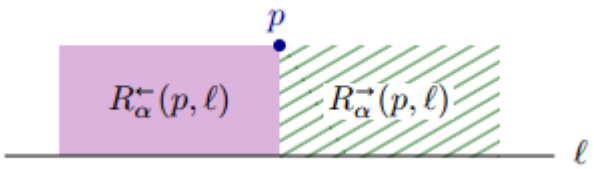
Σχήμα 2.8

2.2 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 $\kappa^*(P, \alpha)$. Ο αλγόριθμος που θα εξετάσουμε είναι ένας 4-προσεγγιστικός αλγόριθμος, δηλαδή έστω $\kappa(P, \alpha)$ η τιμή που επιστρέφει ο αλγόριθμός μας, τότε γνωρίζουμε και θα αποδείξουμε ότι η τιμή ικανοποιεί την παρακάτω συνθήκη

$$\kappa^*(P, \alpha)/4 \leq \kappa(P, \alpha) \leq \kappa^*(P, \alpha)$$

Για μια οριζόντια ευθεία l και ένα σημείο p ∉ l, ορίζουμε ως $R_\alpha^{\rightarrow}(p, l)$ 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 p_x. Παρόμοια, ορίζουμε το ορθογώνιο $R_\alpha^{\leftarrow}(p, l)$, με τη διαφορά ότι περιέχει σημεία με x-συντεταγμένη μικρότερη από p_x. Για μεγαλύτερη πληρότητα δίνεται το Σχήμα 2.6. Επίσης, $R_\alpha(l)$ είναι το σύνολο των ορθογωνίων $\cup_{p \in P} \{ R_\alpha^{\rightarrow}(p, l), R_\alpha^{\leftarrow}(p, l) \}$. 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 $R_\alpha^{\leftarrow}(p, l)$ και $R_\alpha^{\rightarrow}(p, l)$.

Λήμμα 7. *Υπάρχει κάποιο $R \in R_\alpha(l)$ τέτοιο ώστε $|P \cap R| \geq \kappa^*(P, l, \alpha)/4$.*

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 $R^+ := R^* \cap l^+$ και $R^- := R^* \cap l^-$ και υποθέτουμε, χωρίς απώλεια της γενικότητας, ότι $|R^+ \cap P| \geq |R^* \cap P|/2$. Ως p συμβολίζουμε το σημείο στο $P \cap R^*$ με την μέγιστη y-συντεταγμένη. Τότε $R^+ \subset R_\alpha^{\rightarrow}(p, l) \cup R_\alpha^{\leftarrow}(p, l)$. Συνεπώς, τουλάχιστον ένα από τα $R_\alpha^{\rightarrow}(p, l)$, $R_\alpha^{\leftarrow}(p, l)$ πρέπει να περιέχει
$|R^+ \cap P|/2 \geq |R^* \cap P|/2$ σημεία.

Θεώρημα 8. *Δοθέντος ενός συνόλου n σημείων P και μιας τιμής α > 0, μπορούμε να υπολογίσουμε σε χρόνο $O(n\log^2 n)$ τιμή κ(P, α) για την οποία ισχύει*
$\kappa^*(P, \alpha)/4 \leq \kappa(P, \alpha) \leq \kappa^*(P, \alpha)$.

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 $|R \cap P|$ 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

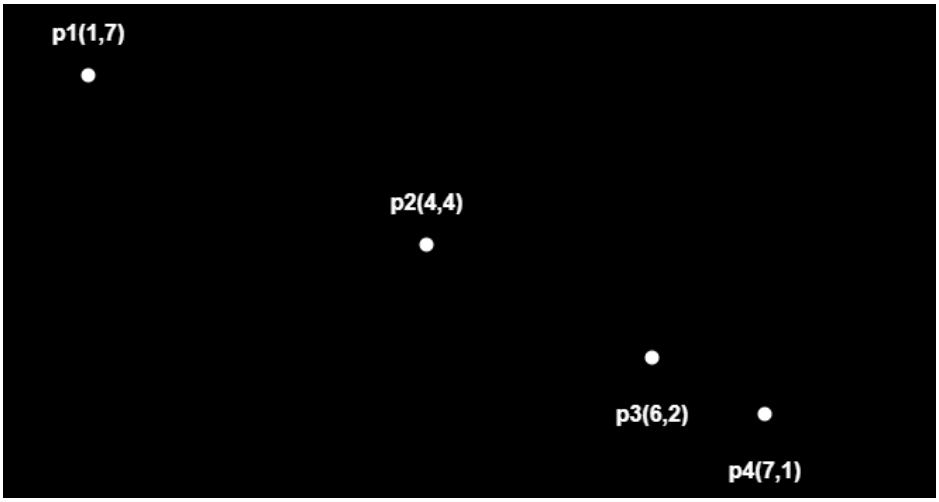
$$\kappa^*(P, \alpha) = \max \{ \kappa^*(P^+, \alpha), \kappa^*(P^-, \alpha), \kappa^*(P, l, \alpha) \}$$

Δημιουργούμε το σύνολο $R_\alpha(l)$ σε χρόνο $O(n)$. Για κάθε ορθογώνιο $R \in R_\alpha(l)$ κάνουμε αναζήτηση στην δομή δεδομένων για να βρούμε την τιμή $|R \cap P|$. Στη συνέχεια, υπολογίζουμε την τιμή $\kappa(P, l, \alpha) = \max \{ |R \cap P| \mid R \in R_\alpha(l) \}$ σε χρόνο $O(n\log n)$. Από το Λήμμα 7 έχουμε $\kappa^*(P, l, \alpha)/4 \leq \kappa(P, l, \alpha) \leq \kappa^*(P, l, \alpha)$. Τότε, επιστρέφουμε την μέγιστη τιμή εκ των $\kappa(P, l, \alpha)$ και $\kappa(P^+, \alpha)$, $\kappa(P^-, \alpha)$ 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 $O(n\log n)$ για συνολικό χρόνο $O(n\log^2 n)$.

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

$$\begin{aligned} \kappa(P, \alpha) &= \max \{ \kappa(P^+, \alpha), \kappa(P^-, \alpha), \kappa(P, l, \alpha) \} \\ &\geq \max \{ \kappa^*(P^+, \alpha)/4, \kappa^*(P^-, \alpha)/4, \kappa^*(P, l, \alpha)/4 \} \\ &= \kappa^*(P, \alpha)/4. \end{aligned}$$

Παρακάτω δίνονται τα βήματα εκτέλεσης του αλγορίθμου για ένα απλό παράδειγμα για εμβαδόν $\alpha = 10$ και σύνολο P των τεσσάρων σημείων όπως φαίνεται στο Σχήμα 2.10.



Σχήμα 2.10

Εν συνεχεία, υπολογίζουμε το σύνολο των σημείων $R_\alpha(l)$, 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 $R_\alpha(l)$ πρέπει να έχει εμβαδό $\alpha = 10$, που είναι και το δοθέν εμβαδόν ορθογωνίου. Θα συμβολίζουμε ένα ορθογώνιο με την μορφή $[x_{low}, x_{high}, y_{low}, y_{high}]$.

- Για το σημείο p_1:
  Βρίσκουμε το $R_\alpha^{\rightarrow}(p_1, l=3)$, για το οποίο δε γνωρίζουμε το x_{high}. Το υπολογίζουμε χρησιμοποιώντας την εξίσωση $\alpha = \beta \cdot \upsilon$, έτσι προκύπτει

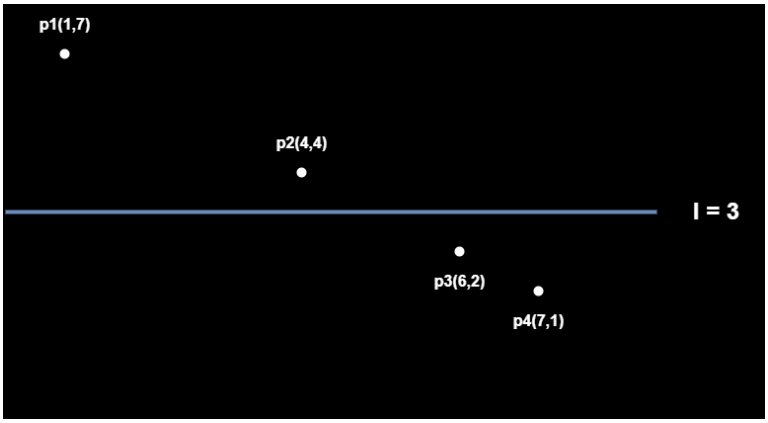
x_{high} = 3.5 και προκύπτει το ορθογώνιο [1, 3.5, 3, 7]. Άρα, $|R_\alpha^{\rightarrow}(p_1, l=3) \cap P| = 1$. Παρόμοια προκύπτει ότι $|R_\alpha^{\leftarrow}(p_1, l=3) \cap P| = 1$.

- Για το σημείο p_2:
  Με παρόμοιο τρόπο υπολογίζουμε τα $R_\alpha^{\rightarrow}(p_2, l=3)$ και $R_\alpha^{\leftarrow}(p_2, l=3)$. Προκύπτει $|R_\alpha^{\rightarrow}(p_2, l=3) \cap P| = 1$ και $|R_\alpha^{\leftarrow}(p_2, l=3) \cap P| = 1$.
- Για το σημείο p_3:
  Με παρόμοιο τρόπο υπολογίζουμε τα $R_\alpha^{\rightarrow}(p_3, l=3)$ και $R_\alpha^{\leftarrow}(p_3, l=3)$. Προκύπτει $|R_\alpha^{\rightarrow}(p_3, l=3) \cap P| = 1$ και $|R_\alpha^{\leftarrow}(p_3, l=3) \cap P| = 1$.
- Για το σημείο p_4:
  Με παρόμοιο τρόπο υπολογίζουμε τα $R_\alpha^{\rightarrow}(p_4, l=3)$ και $R_\alpha^{\leftarrow}(p_4, l=3)$. Προκύπτει $|R_\alpha^{\rightarrow}(p_4, l=3) \cap P| = 1$ και $|R_\alpha^{\leftarrow}(p_4, l=3) \cap P| = 2$. Το ορθογώνιο που καλύπτει δύο σημεία είναι το $[x_{low}, x_{high}, y_{low}, y_{high}] = [2, 7, 1, 3]$.

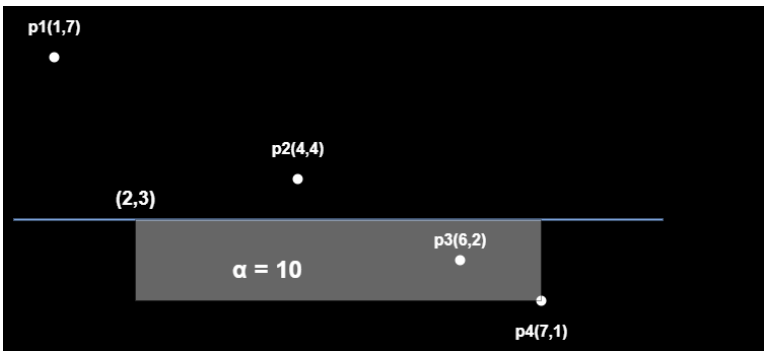
Συνεπώς, έχουμε $\kappa(P, l=3, \alpha=10) = 2$ σημεία.

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

$$[x_{low}, x_{high}, y_{low}, y_{high}] = [2, 7, 1, 3].$$



Σχήμα 2.11



Σχήμα 2.12

Κεφάλαιο 3. Η Υλοποίηση

3.1 Είσοδος – Έξοδος

Η είσοδος και των δύο αλγορίθμων είναι ένα σύνολο σημείων (οποιουδήποτε αριθμού) στον R^2. 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 $[0, 40] \times [0, 40]$ στον R^2, 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:

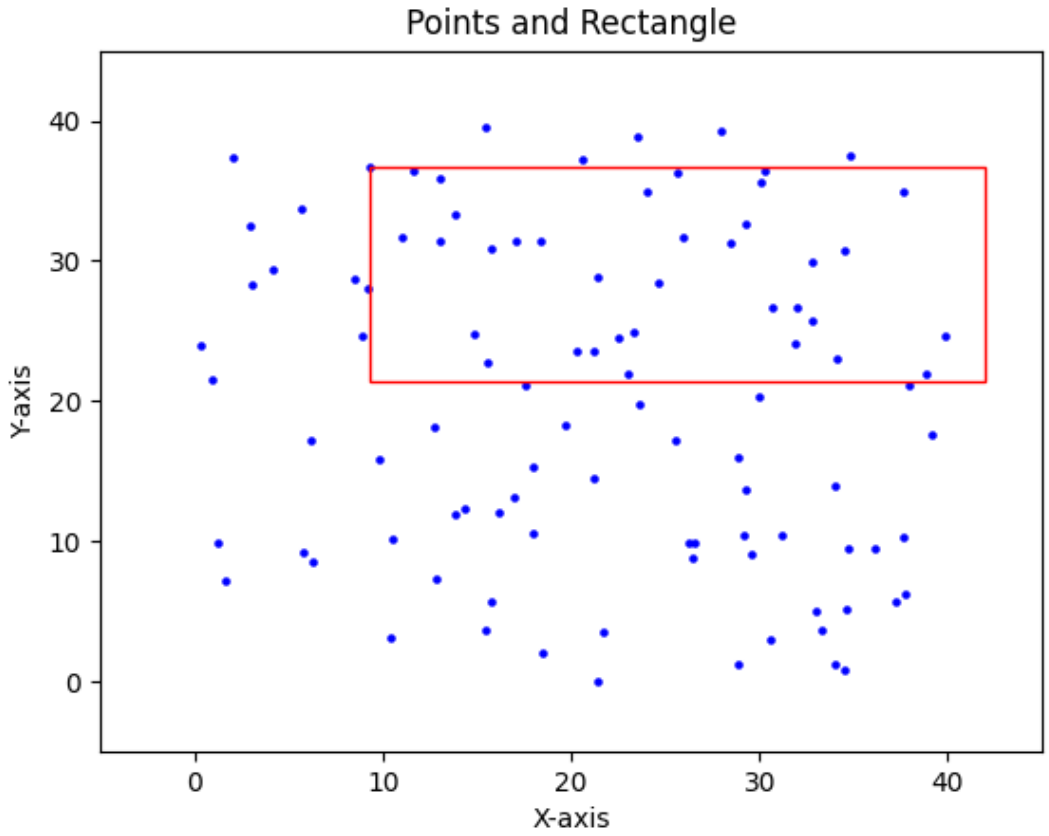
```
P = []

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.

3.2 Δομές Δεδομένων

Η αναπαράσταση όλων των σημείων αλλά και των ορθογωνίων έγινε χρησιμοποιώντας πίνακες της Python. Τα σημεία συμβολίζονται ως $[[x_1, y_1], \dots, [x_n, y_n]]$, ενώ το ορθογώνιο ως $[x_{low}, x_{high}, y_{low}, y_{high}]$, όπου x_{low}, x_{high}, y_{low} και y_{high} 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

3.3 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.

3.3.1 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
```

3.3.2 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
```

3.3.3 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
```

3.3.4 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
```

3.3.5 Η Συνάρτηση compute_best_top_box

Δοθέντος ενός συνόλου Q_i και το σημείο q που είναι το σημείο με την μεγαλύτερη y-συντεταγμένη στο Q_i, 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]
```

3.3.6 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]
```

3.3.7 Η Συνάρτηση lemma3

Η συνάρτηση αυτή υπολογίζει και επιστρέφει το σύνολο Q_p, 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
```

3.3.8 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
```

3.3.9 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 Q_p για κάθε στοιχείο του Q_p 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

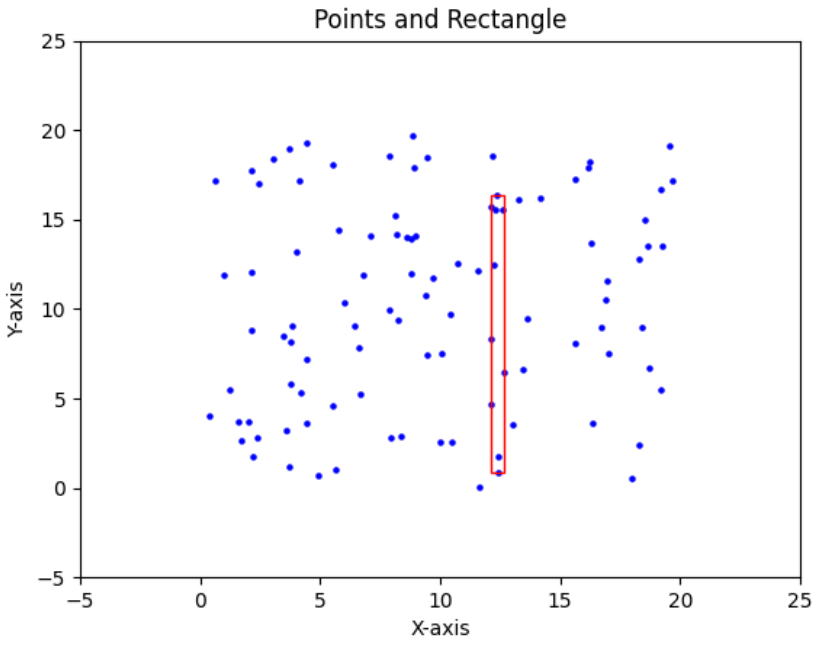
Είναι υπεύθυνη για την αναπαράσταση των σημείων και του ορθογωνίου, με χρήση της βιβλιοθήκης matplotlib.

```
#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')
```

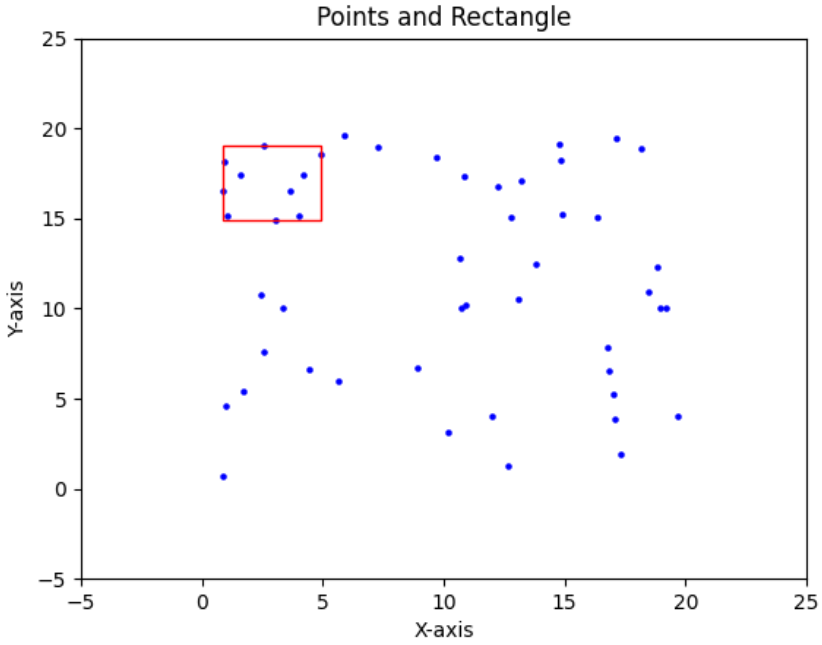
```
# ddd rectangle to the plot
ax.add_patch(rect)

# 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

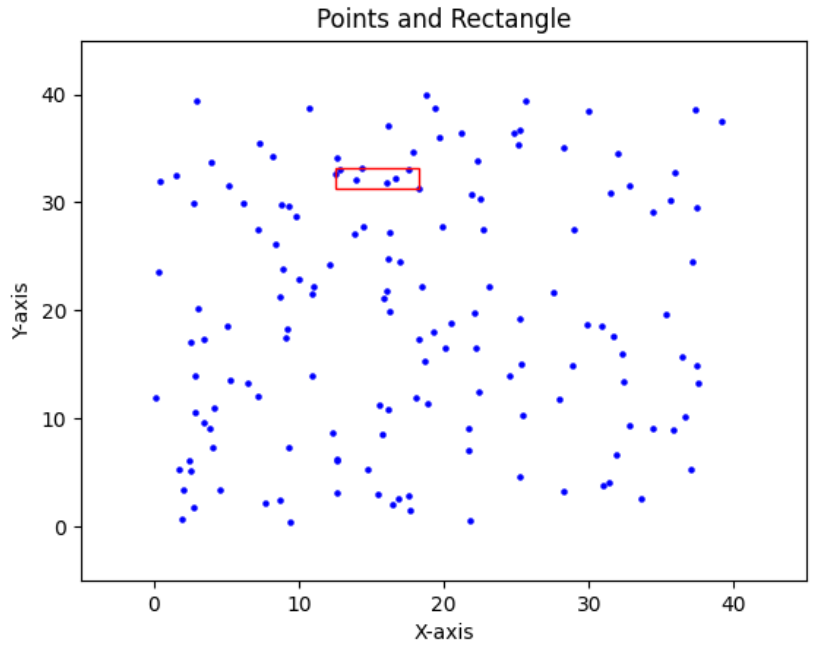
Στα σχήματα Σχήμα 3.2 έως Σχήμα 3.6 φαίνεται η έξοδος του προγράμματος για διάφορες τιμές εισόδου. Στο Σχήμα 3.7 αναπαρίσταται η έξοδος για το παράδειγμα της Ενότητας 2.1.



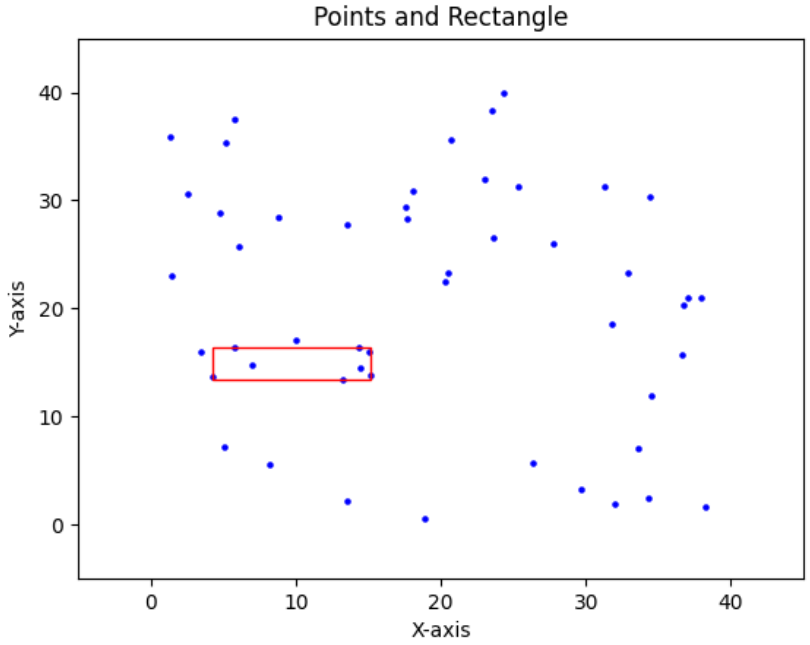
Σχήμα 3.2: coord_range=[0, 20], set_points=100, user_goal_points=10 best_area=8.68



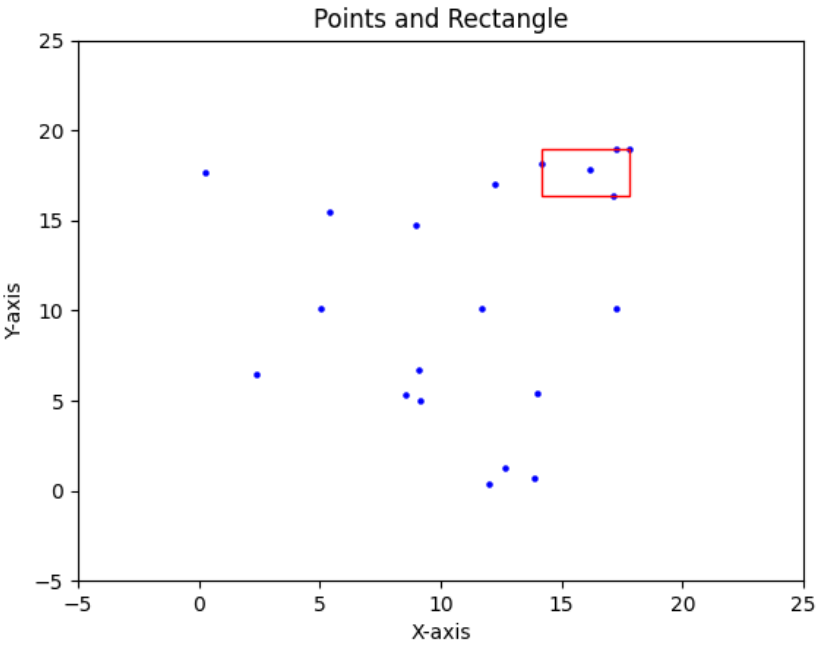
Σχήμα 3.3: coord_range=[0, 20], set_points=50, user_goal_points=10 best_area=16.77



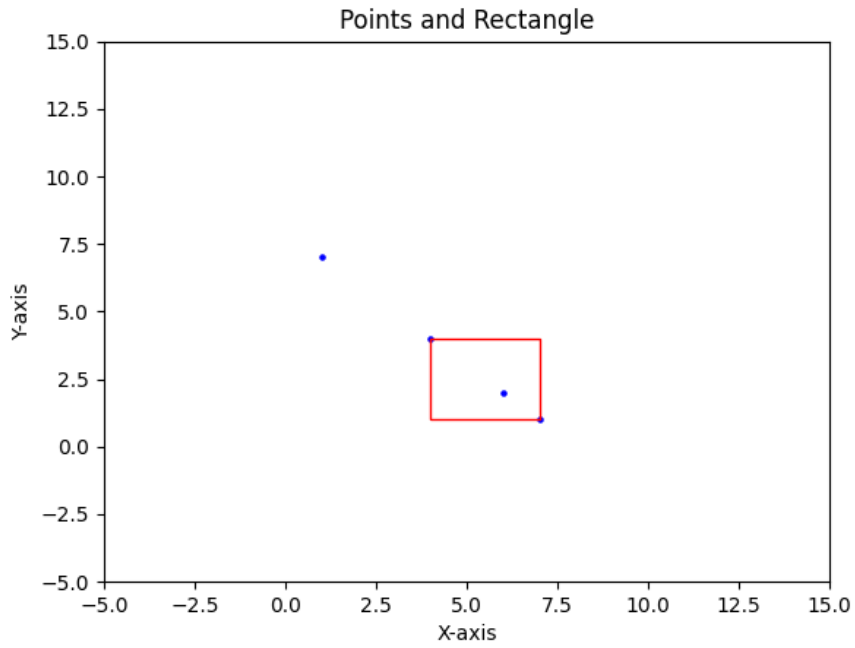
Σχήμα 3.4: coord_range=[0, 40], set_points=150, user_goal_points=8 best_area=10.96



Σχήμα 3.5: coord_range=[0, 40], set_points=50, user_goal_points=8 best_area=10.96



Σχήμα 3.6: coord_range=[0, 20], set_points=20, user_goal_points=5 best_area=9.35



Σχήμα 3.7: Το παράδειγμα της Ενότητας 2.1

3.4 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.

3.4.1 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

$$25 \leq \text{προσεγγιστική\_λύση} \leq 100.$$

```
# 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)
```

3.4.2 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 $R_\alpha(l)$. Αφού είναι γνωστές οι συντεταγμένες κάποιου σημείου $p \in P$, 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
```

3.4.3 Η Συνάρτηση find_max

Επιστρέφει το ζεύγος (μέγιστο_πλήθος_σημείων, ορθογώνιο).

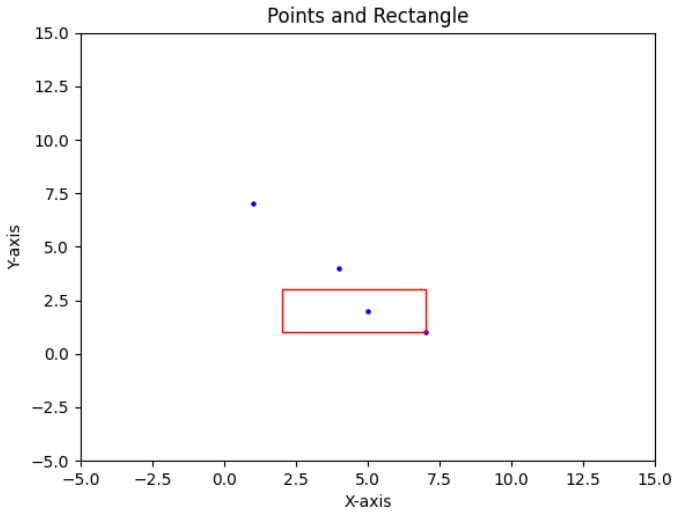
```
def find_max(max_below, max_above, max_current, box_below, box_above,
box_current):
    if (max_below >= max_above) and (max_below >= max_current):
        return max_below, box_below
    elif (max_above >= max_below) and (max_above >= max_current):
        return max_above, box_above
    else:
        return max_current, box_current
```

3.4.4 Η Συνάρτηση plot_points

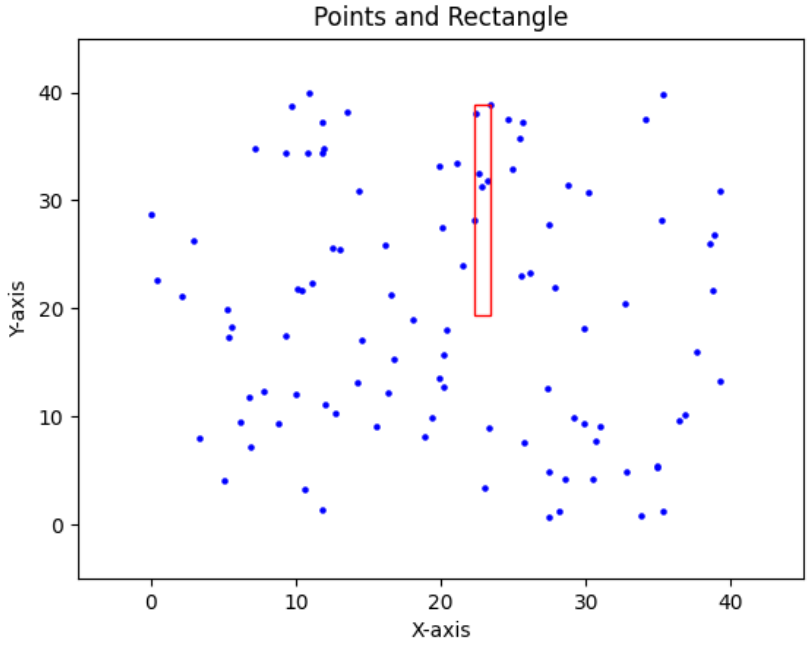
Είναι υπεύθυνη για την αναπαράσταση των σημείων και του ορθογωνίου κάνοντας χρήση της βιβλιοθήκης matplotlib της Python. Ο κώδικας παραλείπεται γιατί είναι ίδιος με αυτόν της Υπό-Ενότητας 3.3.13.

3.4.5 Παραδείγματα Εκτέλεσης του Προγράμματος

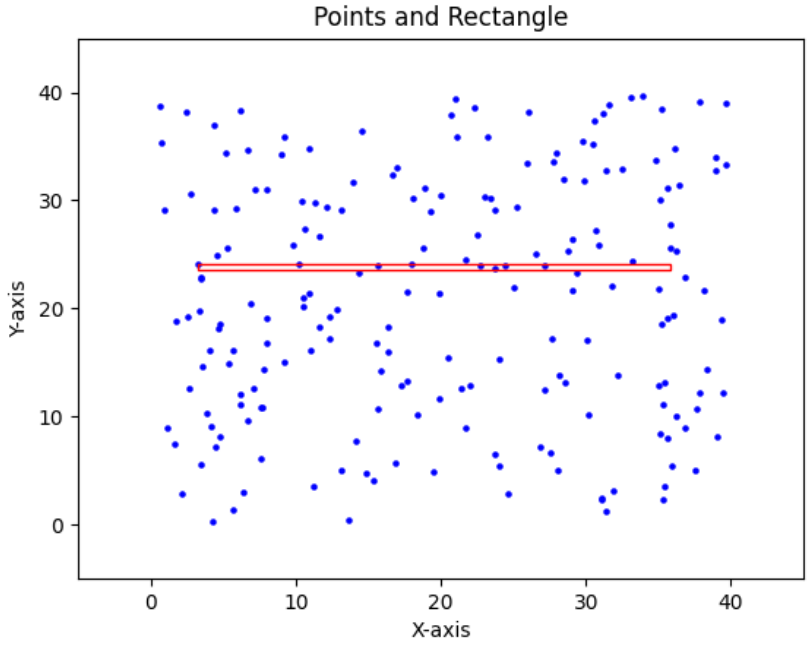
Στα σχήματα Σχήμα 3.9 έως Σχήμα 3.12, φαίνεται η έξοδος του προγράμματος για διάφορες τιμές εισόδου. Στο Σχήμα 3.8 αναπαρίσταται η έξοδος για το παράδειγμα που περιεγράφηκε στην Ενότητα 2.2.



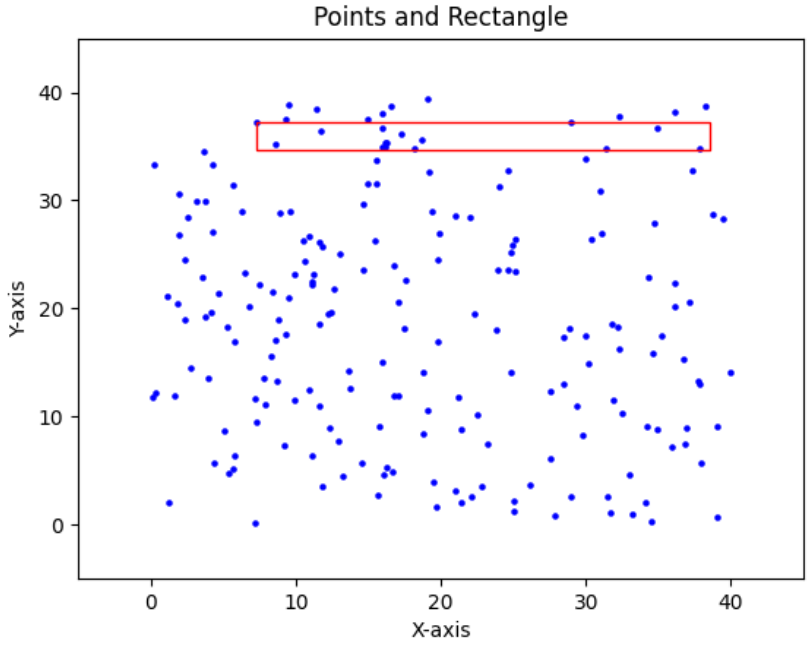
Σχήμα 3.8: Το παράδειγμα της Ενότητας 2.2



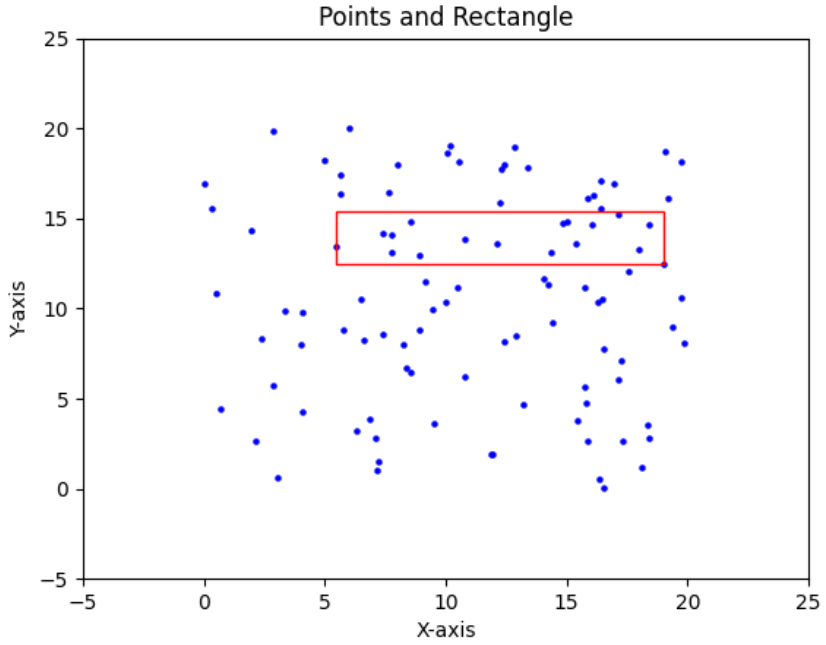
Σχήμα 3.7: coord_range=[0, 40], set_points=100, user_area=20 number_of_points=6



Σχήμα 3.8: coord_range=[0, 40], set_points=200, user_area=20 number_of_points=8



Σχήμα 3.9: coord_range=[0, 40], set_points=200, user_area=80 number_of_points=14



Σχήμα 3.10: coord_range=[0, 20], set_points=100, user_area=40 number_of_points=17

Κεφάλαιο 4. 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 $O(nk^2\log n + n\log^2 n)$ για το πρόβλημα ελαχιστοποίησης εμβαδού ορθογωνίου για δοθέν πλήθος σημείων, ενώ παρουσιάσαμε ένα αλγόριθμο χρόνου $O(n\log^2 n)$ 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.