ΤΜΗΜΑ ΜΗΧ. Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF IOANNINA

Υλοποίηση Αλγορίθμου Φύλαξης x-Μονότονης Πολυγωνικής Γραμμής από Δύο Πύργους

Δημήτριος Βαγενάς

Διπλωματική Εργασία

Επιβλέπων: Λεωνίδας Παληός

Ιωάννινα, Ιούλιος, 2023

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου Λεωνίδα Παληό για την συνεργασία μας, τις συμβουλές και την ηρεμία που μου πρόσφερε για την κατανόηση και την υλοποίηση της εργασίας. Επίσης, θα ήθελα να ευχαριστήσω τους γονείς μου, την γιαγιά μου, την Ευτυχία μου, τις θείες μου Ελένη και Ευαγγελία για την υποστήριξή τους και την υπομονή τους που έδειξαν καθ' όλη τη διάρκεια φοίτησής μου. Τέλος, ευχαριστώ όσους με στήριξαν όσο ήταν ακόμα εδώ.

Δημήτριος Βαγενάς

11/7/2023

Περίληψη

Δεδομένης μίας x-μονότονης πολυγωνικής γραμμής T με n κορυφές, το πρόβλημα των δύο πύργων αναζητά τα δύο κατακόρυφα ευθύγραμμα τμήματα που ονομάζουμε πύργους, των οποίων τα κάτω άκρα (οι βάσεις) βρίσκονται πάνω στη T και τα πάνω άκρα επιτηρούν την T, με την έννοια ότι κάθε σημείο των ακμών της T είναι ορατό από τουλάχιστον ένα πύργο. Υπάρχουν τρεις εκδοχές αυτού του προβλήματος, η διακριτή, η ημι-συνεχής και η συνεχής. Οι δύο πρώτες εκδοχές του προβλήματος, με τις οποίες ασχολούμαστε στην παρούσα εργασία, περιορίζουν την λύση του προβλήματος καθώς στη διακριτή εκδοχή επιβάλλεται και στους δύο πύργους φύλαξης, η βάση τους να βρίσκεται στις κορυφές της T, ενώ στην ημι-συνεχή εκδοχή ο ένας πύργος επιβάλλεται να βρίσκεται σε μία από τις κορυφές της T και ο δεύτερος να μπορεί να βρίσκεται οπουδήποτε στην T. Στη συνεχή περίπτωση οι δύο πύργοι μπορούν να βρίσκονται οπουδήποτε στην T.

Σε αυτή την εργασία μελετήσαμε και υλοποιήσαμε τον αλγόριθμο των Agarwal και άλλων [1] της διακριτής και ημι-συνεχούς εκδοχής φύλαξης x-μονότονης πολυγωνικής γραμμής από δύο πύργους, και οπτικοποιήσαμε τις διαδικασίες των ενδιάμεσων βημάτων που οδηγούν στον υπολογισμό των δύο πύργων που επιτηρούν την T, με το ελάχιστο κοινό ύψος από τις βάσεις τους.

Λέξεις Κλειδιά: Υπολογιστική Γεωμετρία, Αλγόριθμοι Ορατότητας, Φύλαξη Επιπέδου

Abstract

Given a x-monotone polygonal line T with n vertices, the two tower problem searches for the two vertical line segments we call towers, whose lower endpoints (the bases) lie on T and whose top endpoints guard T, in the sense that every point of the edges of T is visible from at least one tower. There are three versions of this problem, discrete, semi-continuous and continuous. The first two versions, which we deal with in this paper, constrain the solution of the problem since in the discrete version both watchtowers have their bases at the vertices of T, while in the semi-continuous version one tower must be located at one of the vertices of T and the second one can lie anywhere in T. In the continuous version the two towers can be anywhere on T.

In this paper we studied and implemented the algorithm by Agarwal et al. [1] the discrete and semi-continuous versions of the two-watchtowers x-monotone polygonal line guarding algorithms and visualized the intermediate procedures leading to the computation of the two watchtowers which guard T with the least common height from their bases.

Keywords: Computational geometry, Visibility algorithms, Terrain guarding

Πίνακας Περιεχομένων

Κεφάλαιο 1. Εισαγωγή

1.1 Βασικοί ορισμοί

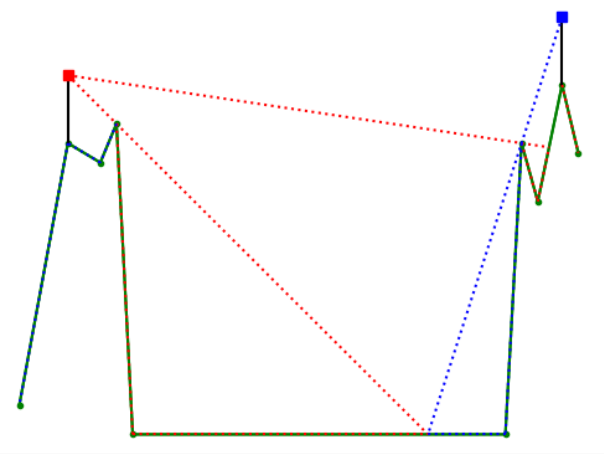
Μία x-μονότονη πολυγωνική γραμμή T είναι μια γραμμή στο επίπεδο που αποτελείται από σημεία με συντεταγμένες (x, y), των οποίων οι τιμές του x αυξάνονται μονότονα. Αυτό σημαίνει ότι όταν προχωράμε από αριστερά προς τα δεξιά στη γραμμή, οι τιμές x των σημείων που συναντάμε αυξάνονται συνεχώς. Ένας πύργος είναι ένα κατακόρυφο ευθύγραμμο τμήμα που αποτελείται από ένα κατώτερο άκρο (βάση) που είναι ένα σημείο που ανήκει στην T, το άνω άκρο το οποίο επιτηρεί την T και το ύψος $h \geq 0$ που δηλώνει πόσο ψηλά είναι το άνω άκρο του πύργου από την βάση. Λέμε πως ένα σημείο $t \in T$ είναι ορατό (φυλάσσεται) από έναν πύργο w εάν το άνω άκρο του w «βλέπει» το t, δηλαδή όταν το ευθύγραμμο τμήμα που ενώνει το t με το ανώτατο άκρο του w βρίσκεται πάνω από την T. Δύο πύργοι που είναι τοποθετημένοι πάνω στη T επιτηρούν την T (ή δημιουργούν ζεύγος φύλαξης της T ή εποπτεύουν ή «βλέπουν» την T) όταν κάθε σημείο της T είναι ορατό από το άνω άκρο τουλάχιστον ενός από τους δύο πύργους.

1.2 Αντικείμενο της Διπλωματικής Εργασίας

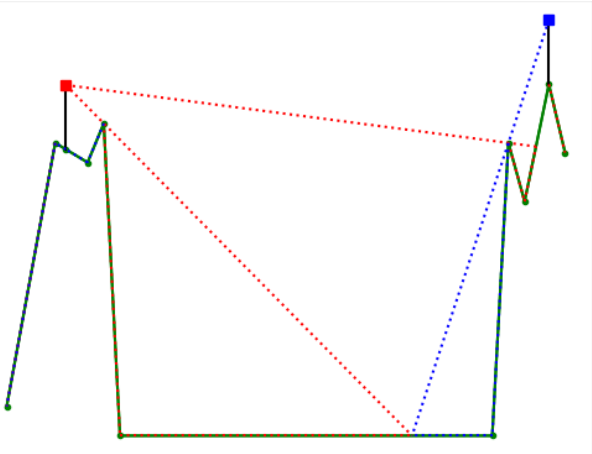
Σε αυτή την εργασία μελετήσαμε, υλοποιήσαμε και οπτικοποιήσαμε τον αλγόριθμο φύλαξης x-μονότονης πολυγωνικής γραμμής από δύο πύργους όπως περιγράφεται από τον Agarwal και άλλους [1] και ορίζεται ως εξής. Δεδομένης μίας x-μονότονης πολυγωνικής γραμμής T με n ακμές, υπολόγισε το ελάχιστο κοινό ύψος $h \geq 0$ για το οποίο υπάρχουν δύο σημεία $u_1, u_2 \in T$ στα οποία αν υψωθούν δύο πύργοι με ύψος h τότε οι δύο πύργοι επιτηρούν την T.

Υπάρχουν τρεις εκδόσεις του παραπάνω προβλήματος. Η διακριτή (Σχήμα 1.1), η οποία περιορίζει την λύση του προβλήματος καθώς υποχρεώνει τους πύργους να υψώνονται μόνο σε κορυφές της T. Η ημι-συνεχής (Σχήμα 1.2), που και αυτή περιορίζει την λύση του προβλήματος, καθώς υποχρεώνει τον πρώτο πύργο να υψωθεί σε κορυφή της T, ενώ ο δεύτερος μπορεί να υψωθεί οπουδήποτε στη T. Τέλος, η συνεχής εκδοχή δεν έχει κάποιο περιορισμό, διότι σε αυτή την περίπτωση οι δύο πύργοι μπορούν να είναι υψωμένοι οπουδήποτε στην T. Εμείς ασχοληθήκαμε με τις δύο πρώτες εκδόσεις του προβλήματος για τις οποίες υλοποιήσαμε και οπτικοποιήσαμε τις ενδιάμεσες διαδικασίες που παράγουν την λύση. Η αλγοριθμική πολυπλοκότητα της διακριτής και ημι-συνεχούς εκδοχής όπως περιγράφεται στον αλγόριθμο [1] είναι $O(n^2 \log^4 n)$. Εμείς, κατασκευάσαμε έναν κώδικα που ακολουθεί σχεδόν κατά γράμμα τον αλγόριθμο [1], επειδή επιλέξαμε να υλοποιήσουμε έναν υπολογισμό με διαφορετικό τρόπο για

λόγους απλότητας (όπως περιγράφουμε στην ενότητα 2.1 Θεωρητικό Υπόβαθρο) επιτυγχάνοντας έτσι πολυπλοκότητα $O(n^3 \log^2 n)$.



Σχήμα 1.1 *Διακριτή λύση προβλήματος.*



Σχήμα 1.2 *Ημι-συνεχής λύση προβλήματος.*

1.3 Σχετικά Ερευνητικά Αποτελέσματα

Η φύλαξη επιπέδου από πύργους είναι μία ειδική περίπτωση της γενικής κατηγορίας προβλημάτων ορατότητας στις δύο και τρεις διαστάσεις, γνωστά και ως προβλήματα πινακοθήκης τα οποία έχουν μελετηθεί εκτενώς για πάνω από τρεις δεκαετίες. Τα προβλήματα αυτά έχουν πολυάριθμες εφαρμογές στην επιτήρηση, στην πλοήγηση, στην υπολογιστική όραση, την μοντελοποίηση και τα γραφικά, το σύστημα γεωγραφικών πληροφοριών (G.I.S.) και πολλά άλλα. Βλέπε [2] για μια έρευνα των προβλημάτων πινακοθήκης.

Το πρόβλημα της φύλαξης στο R^2 από δύο πύργους έχει μελετηθεί σε αρκετές εργασίες. Πέρα τον αλγόριθμο των Agarwal και άλλων [1], ο Bespamyatnikh και άλλοι [3] έδειξαν πως το πρόβλημα, εάν υπάρχουν δύο πύργοι με δεδομένο ύψος h που μπορούν να φυλάξουν μία T, μπορεί να επιτευχθεί σε χρόνο $O(n^3)$. Χρησιμοποιώντας παραμετρική αναζήτηση (parametric search), για το πρόβλημα της βελτιστοποίησης επιτυγχάνουν χρόνο $O(n^3 \log^2 n)$. Επίσης παρουσιάζουν έναν αλγόριθμο $O(n^4)$ που αποφεύγει την παραμετρική αναζήτηση. Ακόμα, οι Ben-Mosche και άλλοι [4] παρουσίασαν τρόπο επίλυσης της διακριτής εκδοχής του προβλήματος σε αλγοριθμικό χρόνο $O(n^{2.688} \log^2 n)$ με χρήση παραμετρικής αναζήτησης.

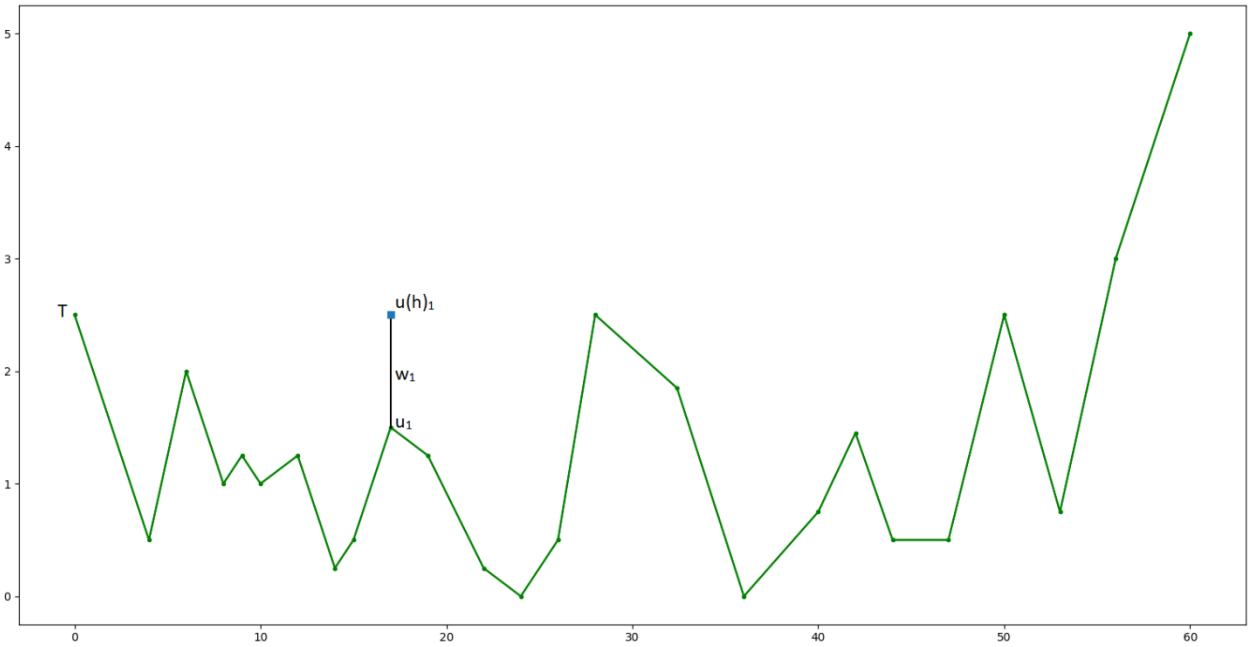
1.4 Δομή της Διπλωματικής Εργασίας

Ο αλγόριθμος [1] που μελετήσαμε, επιλύει το πρόβλημα φύλαξης x-μονότονης πολυγωνικής γραμμής από δύο πύργους με αλγοριθμική πολυπλοκότητα $O(n^2 \log^4 n)$. Αυτό είναι αποτέλεσμα εφαρμογής μεθόδων που αποφασίζουν ταχύτερα από άλλες ποιο είναι το ζεύγος πύργων που έχει το ελάχιστο κοινό ύψος το οποίο επιτηρεί την T. Στις επόμενες ενότητες περιγράφουμε σε βάθος αυτές τις μεθόδους και αναλύουμε το θεωρητικό υπόβαθρο που υλοποιήσαμε για να επιτευχθεί αυτή η χαμηλή πολυπλοκότητα. Εξηγούμε πώς συνδυάζουμε τις διαδικασίες ώστε να παραχθεί ο τελικός αλγόριθμος και δείχνουμε ένα παράδειγμα εφαρμογής του. Τέλος, παρουσιάζουμε αναλυτικά τον κώδικα του αλγορίθμου όπως τον υλοποιήσαμε στην γλώσσα προγραμματισμού Python και κάποια παραδείγματα εκτέλεσης του προγράμματος.

Κεφάλαιο 2. Ο Αλγόριθμος Φύλαξης

2.1 Θεωρητικό Υπόβαθρο

Θέτουμε T μία x-μονότονη πολυγωνική γραμμή στο R^2, με n ακμές, και V το σύνολο των κορυφών της T. Ένας πύργος έχει βάση ένα σημείο $u_1 \in V$. Το άνω άκρο που επιτηρεί την T βρίσκεται σε ύψος $h_1 \geq 0$ πάνω από το u_1 και το συμβολίζουμε με $u(h)_1$. Συνολικά, το κατακόρυφο ευθύγραμμο τμήμα που αποτελεί τον πύργο το συμβολίζουμε με w_1 (Σχήμα 2.1).



Σχήμα 2.1 *Η x-μονότονη πολυγωνική γραμμή T και ένας πύργος w_1 με βάση u_1 και ύψος h_1*

2.1.1 Σύνολο P

Το σύνολο P αποτελείται από όλες τις μη ορατές κορυφές και τα τελικά σημεία των μερικώς ορατών ακμών απ' όπου ξεκινάει το μη ορατό μέρος αυτών των ακμών.

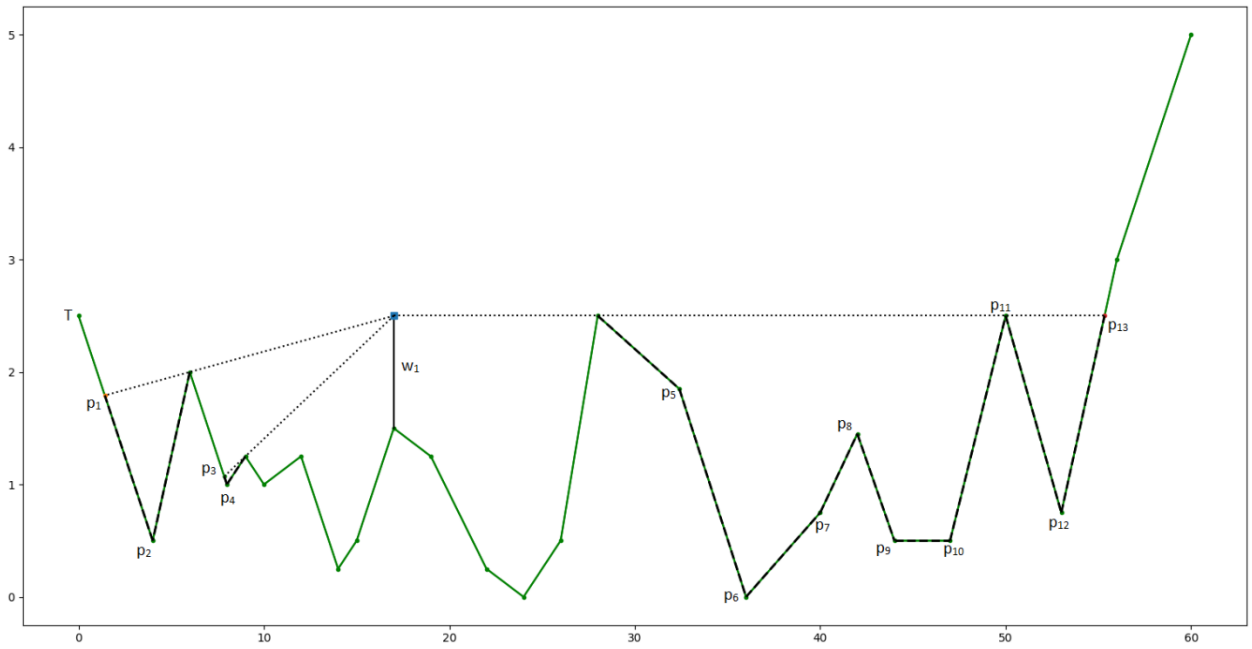
Για να υπολογίσουμε τα μη ορατά μέρη της T κατασκευάζουμε ένα σύνολο P που περιέχει όλες τις κορυφές που δεν είναι ορατές από τον πύργο w_1 και σημεία ακμών που χάνεται η ορατότητα. Η κορυφή στην οποία χάνεται η ορατότητα και ξεκινάει ένα μη ορατό τμήμα δεν περιέχεται στο σύνολο P, όμως το σημείο που τελειώνει το μη ορατό τμήμα είτε αυτό είναι

κορυφή, είτε είναι σημείο πάνω σε μία ακμή, το υπολογίζουμε και το συμπεριλαμβάνουμε στο σύνολο P. Το σύνολο P, το υπολογίζουμε σε γραμμικό χρόνο με την εξής τεχνική:

Για τον πύργο w_1 έχουμε ένα περιστροφικό ευθύγραμμο τμήμα e, του οποίου το ένα άκρο είναι το $u(h)_1$. Αρχικά ελέγχουμε την ορατότητα αριστερά του πύργου, οπότε το άλλο άκρο του περιστροφικού ευθύγραμμου τμήματος είναι η επόμενη κορυφή v_i αριστερά του πύργου w_1. Έτσι αρχικοποιούμε το τωρινό ευθύγραμμο τμήμα ως cur_e = ($u(h)_1$, v_i) και την κλίση του min_s (!= ∞ επειδή η T είναι x-μονότονη πολυγωνική γραμμή). Στη συνέχεια ξεκινάμε μία επαναληπτική διαδικασία που ελέγχει την τωρινή ελάχιστη κλίση του cur_e με την κλίση του επόμενου τμήματος next_e = ($u(h)_1$, v_{i-1}). Αν η κλίση του next_e είναι μικρότερη ή ίση, τότε θέτουμε ως τωρινό ευθύγραμμο τμήμα το επόμενο ευθύγραμμο τμήμα (cur_e = next_e) και τωρινή ελάχιστη κλίση την κλίση του επόμενου τμήματος (min_s = next_s) και συνεχίζουμε στο επόμενο ευθύγραμμο τμήμα μειώνοντας τον μετρητή i κατά 1. Διαφορετικά, αν η κλίση είναι μεγαλύτερη αυτό σημαίνει πως ξεκινάει ένα τμήμα της T που δεν είναι ορατό από τον w_1. Για κάθε αριστερότερη κορυφή που δημιουργεί μεγαλύτερη ή ίση κλίση από την τωρινή ελάχιστη κλίση next_s ≥ min_s, προσθέτουμε την v_i στο σύνολο P, διατηρώντας σε ένα άλλο σύνολο και την κορυφή από την οποία ξεκινάει η απώλεια ορατότητας για να καταγράψουμε όλα τα «σκοτεινά» σημεία της T. Μόλις η επόμενη κορυφή που εξετάζουμε δημιουργεί ευθύγραμμο τμήμα (next_e) με κλίση μικρότερη της ελάχιστης κλίσης (next_s<min_s), ελέγχουμε πού τέμνει ο φορέας του ευθύγραμμου τμήματος που διατηρούσε τη μέχρι πρότινος ελάχιστη κλίση στην ακμή a που αποτελείται από την τελευταία κορυφή που ελέγξαμε και την προηγούμενη αυτής (a = (v_i, v_{i-1})). Αυτή η τομή κεα είναι το σημείο που αρχίζει να χάνεται η ορατότητα της μερικώς ορατής ακμής a από τον w_1. Το σημείο τομής k το προσθέτουμε στο σύνολο P. Η διαδικασία για το τμήμα της T αριστερά του w_1 ολοκληρώνεται όταν εξετάσουμε και το ευθύγραμμο τμήμα next_e που δημιουργείται από την αριστερότερη κορυφή της T. Όλα τα στοιχεία που συλλέξαμε με την παραπάνω διαδικασία, τα συλλέξαμε με φθίνουσα x διάταξη και για αυτό στο τέλος τα καθρεπτίζουμε (mirror) στον άξονα x έτσι ώστε να είναι ταξινομημένα με αύξουσα x διάταξη.

Για το υπολογισμό της ορατότητας και του συνόλου P στο τμήμα της T δεξιά του w_1, ο αλγόριθμος είναι ίδιος με την διαφορά ότι η επόμενη κορυφή v_i είναι η αμέσως δεξιότερη και συνεπώς ο μετρητής i αντί να μειώνεται, αυξάνεται κατά 1 και το ευθύγραμμο τμήμα που διατηρούμε είναι αυτό με την μεγαλύτερη κλίση (max_s αντί για min_s). Τις κορυφές που ανήκουν στα «σκοτεινά» τμήματα της T τις βρίσκουμε όταν τα next_e έχουν next_s ≥ max_s.

Τέλος, τα στοιχεία έχουν προσαρτηθεί (append) με την επιθυμητή σειρά (αύξουσα κατά x συντεταγμένη) και έχουμε υπολογίσει το σύνολο P αλλά και όλες τις μη ορατές ακμές ή μη ορατά τμήματα ακμών που μπορεί να υπάρχουν όπως φαίνεται στο Σχήμα 2.2 σε χρόνο O(n).



Σχήμα 2.2 *Το μη ορατό τμήμα της T από τον w_1 και τα σημεία του συνόλου P.*

2.1.2 Αριστερά ζεύγη ορατότητας

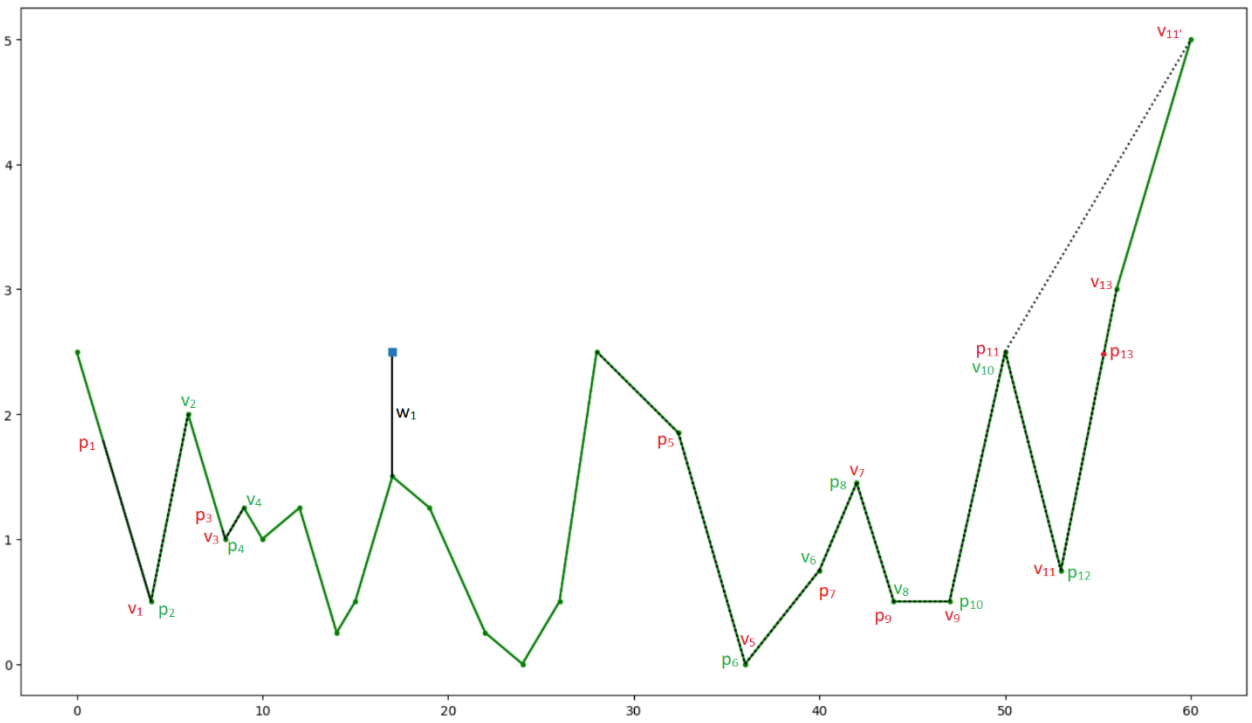
Τα αριστερά ζεύγη ορατότητας είναι ευθύγραμμα τμήματα των οποίων οι φορείς ορίζουν τα σημεία που πρέπει να υψωθεί ο συμπληρωματικός πύργος w_2, δεξιά του πρώτου, ώστε να είναι ορατά όλα τα σημεία του συνόλου P.

Τα σημεία του συνόλου P τα χρειαζόμαστε για να κατασκευάσουμε τα αριστερά ζεύγη ορατότητας τα οποία ορίζουμε ως το σύνολο Π(P, V) και καθορίζουν τον υπολογισμό του συμπληρωματικού πύργου. Θεωρώντας v μία κορυφή ώστε vϵV, λέμε πως ένα σημείο pϵP σχηματίζει «αριστερό» ζεύγος ορατότητας με τη v εάν:

i) Το p βρίσκεται αριστερά του v

ii) Το p «βλέπει» το v

iii) Το p είναι το δεξιότερο σημείο στο σύνολο P που ικανοποιεί τις i) και ii). Η φυσική σημασία αυτής της συνθήκης είναι ότι το ζεύγος pv είναι το ζεύγος που σχηματίζει μεγαλύτερη κλίση.



Σχήμα 2.3 *Τα αριστερά ζεύγη ορατότητας που δημιουργούνται από τον πύργο* w_1.

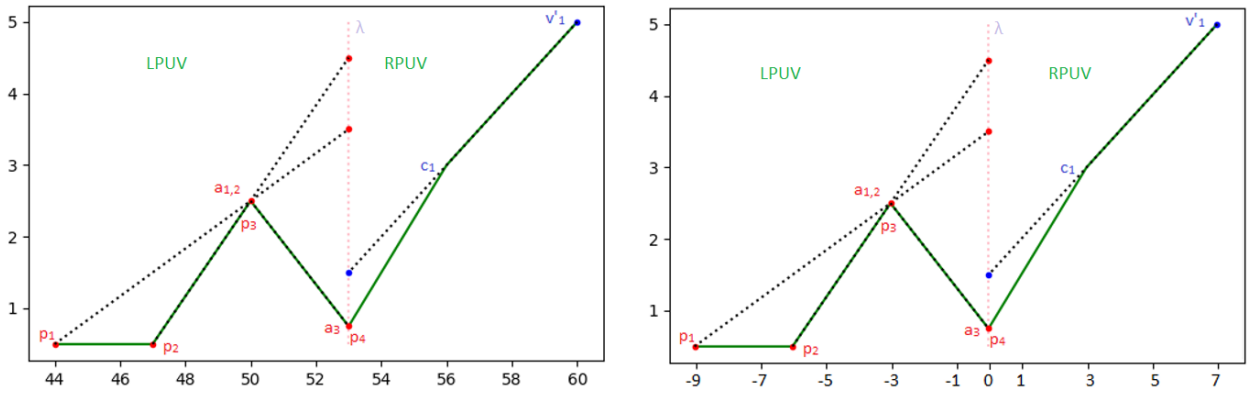
Για να υπολογίσουμε τα αριστερά ζεύγη ορατότητας με την επιθυμητή πολυπλοκότητα εφαρμόζουμε την παρακάτω διαίρει και βασίλευε τεχνική:

Αφού έχουμε υπολογίσει το σύνολο P, ορίζουμε το σύνολο (terrain with P) TWP = P U V. «Διαιρούμε» ακέραια το σύνολο TWP με την βοήθεια μίας κατακόρυφης ευθείας λ στα δύο, δηλαδή παράγουμε ένα αριστερό και ένα δεξί υποσύνολο. Αυτή τη διαίρεση την εφαρμόζουμε αναδρομικά έως ότου ο πληθάριθμος π των σημείων pϵP και vϵV (επειδή μπορεί ένα σημείο pϵV, ελέγχουμε την περίπτωση έτσι ώστε να μην το προσμετρήσουμε διπλά στο π) του παραγόμενου υποσυνόλου να είναι σχετικά μικρός π.χ. π ≤ 5), ώστε να μπορούμε γρήγορα να εξάγουμε τα αριστερά ζεύγη ορατότητας από το αριστερό υποσύνολο LPUV και δεξί υποσύνολο RPUV. Τα αριστερά ζεύγη για το LPUV στην μικρότερη εκδοχή του (π ≤ 5) τα υπολογίζουμε ελέγχοντας για κάθε κορυφή v_iϵLPUV από τα δεξιά προς τα αριστερά, ποιο από τα σημεία p, τέτοιο ώστε pϵP και pϵLPUV, εάν υπάρχει, είναι το δεξιότερο, είναι ορατό και βρίσκεται αριστερά από την κορυφή v_i. Εφαρμόζοντας τον ίδιο αλγόριθμο στα μικρότερα αριστερά και δεξιά υποσύνολα LPUV, RPUV, εξάγουμε τα αριστερά ζεύγη από αυτά και τα συμβολίζουμε ως Π(PL, VL) και Π(PR, VR) αντίστοιχα. Ως PL και PR έχουμε τα σημεία pϵP αριστερά και δεξιά της κατακόρυφης γραμμής λ αντίστοιχα και ως VL και VR έχουμε τις κορυφές vϵV αριστερά και δεξιά της κατακόρυφης ευθείας λ αντίστοιχα.

Μέχρι στιγμής έχουμε υπολογίσει τα αριστερά ζεύγη ορατότητας Π(PL, VL) και Π(PR, VR), όμως δεν έχουμε εξετάσει αν κάποια κορυφή v_r ϵ VR που δεν έκανε αριστερό ζεύγος στο Π(PR, VR), μπορεί να σχηματίσει αριστερό ζεύγος με κάποιο σημείο p_L ϵ PL. Αυτά τα αριστερά ζεύγη ορατότητας τα συμβολίζουμε ως Π(PL, VR'), με VR' σύνολο από κορυφές vϵVR που δεν δημιούργησαν κάποιο ζεύγος στο σύνολο Π(PR, VR). Έτσι καταλήγουμε πως:

$$\Pi(P,V) = \Pi(PL, VL) \cup \Pi(PR, VR) \cup \Pi(PL, VR')$$

Στη γενική περίπτωση, για τον υπολογισμό του συνόλου Π(PL, VR'), πρέπει να κάνουμε έναν μετασχηματισμό μεταφοράς στον άξονα x σε όλα μας τα δεδομένα έτσι ώστε η κατακόρυφη ευθεία λ να ταυτίζεται με τον άξονα y. Με αυτόν το μετασχηματισμό, τα σημεία του συνόλου LPUV έχουν τετμημένες (x συντεταγμένη) μικρότερες ή ίσες του μηδενός, ενώ τα σημεία του συνόλου RPUV έχουν τετμημένες μεγαλύτερες ή ίσες του μηδενός. Για κάθε σημείο pϵPL θέλουμε να βρούμε την ευθεία που πηγάζει από το p, έχει την μικρότερη κλίση και «βλέπει» τη λ. Δηλαδή, δεν τέμνει καμία ακμή της T, αλλά διέρχεται από μία κορυφή aϵV που της επιτρέπει να δει τη λ από την τομή του φορέα του ευθύγραμμου τμήματος pa = (p,a) με την λ και πάνω. Αφού έχουμε υπολογίσει το σύνολο VR', για κάθε v'ϵVR' εφαρμόζουμε παρόμοια διαδικασία βρίσκοντας τώρα την κορυφή cϵV που σχηματίζει με την v' ένα ευθύγραμμο τμήμα v'c = (v',c), που ο φορέας του δεν τέμνει καμία ακμή της T και έχει την μεγαλύτερη κλίση, έτσι ώστε το v' να «βλέπει» την λ από το σημείο τομής της λ με το φορέα του v'c και πάνω (Σχήμα 2.4).



Σχήμα 2.4 *Ο υπολογισμός των τμημάτων pa και v'c και ο μετασχηματισμός των τετμημένων τμήματος της T. Σημείωση: Το σημείο p που είναι πάνω στην λ (p_4) ανήκει στο RPUV και δημιουργεί ζεύγος με την κορυφή c_1. Για αυτό η κορυφή c_1 δεν ανήκει στο σύνολο VR'.*

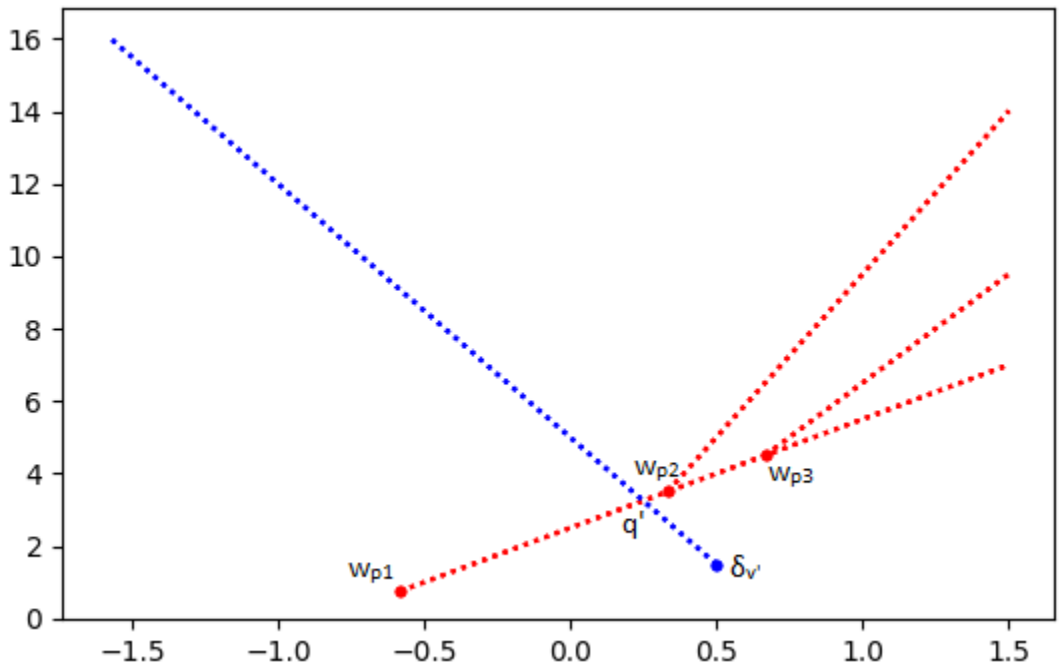
Στο δυικό επίπεδο, η δυική εικόνα ενός σημείου (k, l), είναι η ευθεία y = -kx + l, και η δυική εικόνα μίας ευθείας με εξίσωση y = mx + n είναι το σημείο (m,n). Για απλότητα, κάποιος

θα μπορούσε να πει πως στο δυικό μετασχηματισμό οι ευθείες μετατρέπονται σε σημεία και τα σημεία σε ευθείες, σύμφωνα με τους παραπάνω μετασχηματισμούς.

Έχοντας υπολογίσει για κάθε pϵPL την εξίσωση του φορέα του ευθύγραμμου τμήματος pa και για κάθε v'ϵVR' την εξίσωση του φορέα της v'c, είμαστε έτοιμοι να περάσουμε στο δυικό επίπεδο. Ας ορίσουμε την ευθεία e_p που δημιουργείται με το δυικό μετασχηματισμό ενός σημείου p. Η γεωμετρική σημασία των δυικών εικόνων όλων των σημείων που απαρτίζουν την ευθεία e_p είναι οι άπειρες ευθείες που διέρχονται από το σημείο p. Όμως, εάν θέσουμε μία ημι-ευθεία w_p που βρίσκεται στον φορέα της e_p η οποία πηγάζει από την δυική εικόνα της εξίσωσης του φορέα της pa και κινείται προς τα δεξιά, ο γεωμετρικός τόπος αυτής της ημι-ευθείας είναι όλες οι ευθείες που διέρχονται από το p και κινούνται δεξιά του p τέμνοντας την λ χωρίς να τέμνουν καμία ακμή της T. Όλες οι w_p ημι-ευθείες δεν τέμνονται μεταξύ τους με την έννοια ότι η μία δεν διαπερνάει την άλλη, όμως μπορεί ένα ή περισσότερα σημεία p να έχουν κοινό a το οποίο είναι ταυτόχρονα και σημείο p και αυτό έχει ως αποτέλεσμα οι ημι-ευθείες που έχουν κοινό a-p σημείο να ξεκινούν από την ημι-ευθεία που δημιουργείται από αυτό το p-a σημείο.

Σε πλήρη αναλογία, για κάθε v'ϵVR' κατασκευάζουμε την ημι-ευθεία $\delta_{v'}$ που ο φορέας της είναι η δυική εικόνα του σημείου v', το αρχικό σημείο η δυική εικόνα της εξίσωσης του φορέα του ευθύγραμμου τμήματος v'c και η φορά της προς τα αριστερά.

Ο λόγος που υπολογίσαμε τις ημι-ευθείες w_p και $\delta_{v'}$ είναι ότι η εάν κάποια $\delta_{v'}$ τέμνει κάποια από τις ημι-ευθείες w_p, τότε αυτό σημαίνει ότι το σημείο v' είναι ορατό από το σημείο p και συγκεκριμένα εάν η ημι-ευθεία w_p είναι η πρώτη που τέμνει η ημι-ευθεία $\delta_{v'}$, τότε αυτό σημαίνει ότι το σημείο p και η κορυφή v' δημιουργούν αριστερό ζεύγος ορατότητας, δηλαδή το ζεύγος (p, v')ϵ Π(PL, VR'). Ο υπολογισμός του ζεύγους ιδανικά θα γινόταν με την τεχνική rayshooting [1], αλλά εμείς για λόγους απλότητας επιλέξαμε να το υλοποιήσουμε με έναν πιο αργό αλγόριθμο, υπολογίζοντας όλες τις τομές q μίας ημι-ευθείας $\delta_{v'}$ με τις ημι-ευθείες w_p, αν αυτές υπάρχουν, και όποιο q' είναι πιο κοντά στο αρχικό σημείο της $\delta_{v'}$ τότε, το p σημείο από το οποίο δημιουργήθηκε η w_p ημι-ευθεία που τέμνεται από την $\delta_{v'}$ στο σημείο q' λέμε ότι δημιουργεί ζεύγος ορατότητας με την κορυφή v', (p, v')ϵ Π(PL, VR') (Σχήμα 2.5).



Σχήμα 2.5 *Οι ημι-ευθείες w_p και $\delta_{v'}$ στο δυικό επίπεδο και η τομή τους q' που υποδηλώνει αριστερό ζεύγος ορατότητας.*

Τα αριστερά ζεύγη ορατότητας τα υπολογίζουμε τελικά σε χρόνο $O(n^2 \log n)$.

2.1.3 Δεξιά ζεύγη ορατότητας

Τα δεξιά ζεύγη ορατότητας είναι ευθύγραμμα τμήματα των οποίων οι φορείς ορίζουν τα σημεία που πρέπει να υψωθεί ο συμπληρωματικός πύργος w_2, αριστερά του πρώτου, ώστε να είναι ορατά όλα τα σημεία του συνόλου P.

Για τον υπολογισμό των πύργων που επιτηρούν την T στη διακριτή εκδοχή, χρειαζόμαστε μόνο τα «αριστερά» ζεύγη ορατότητας. Στην ημι-συνεχή εκδοχή όμως χρειαζόμαστε και τα «δεξιά» ζεύγη ορατότητας (Σχήμα 2.6). Θεωρώντας v μία κορυφή ώστε $v \in V$, λέμε πως ένα σημείο $p \in P$ σχηματίζει «δεξί» ζεύγος ορατότητας με τη v εάν:

i) Το p βρίσκεται δεξιά του v

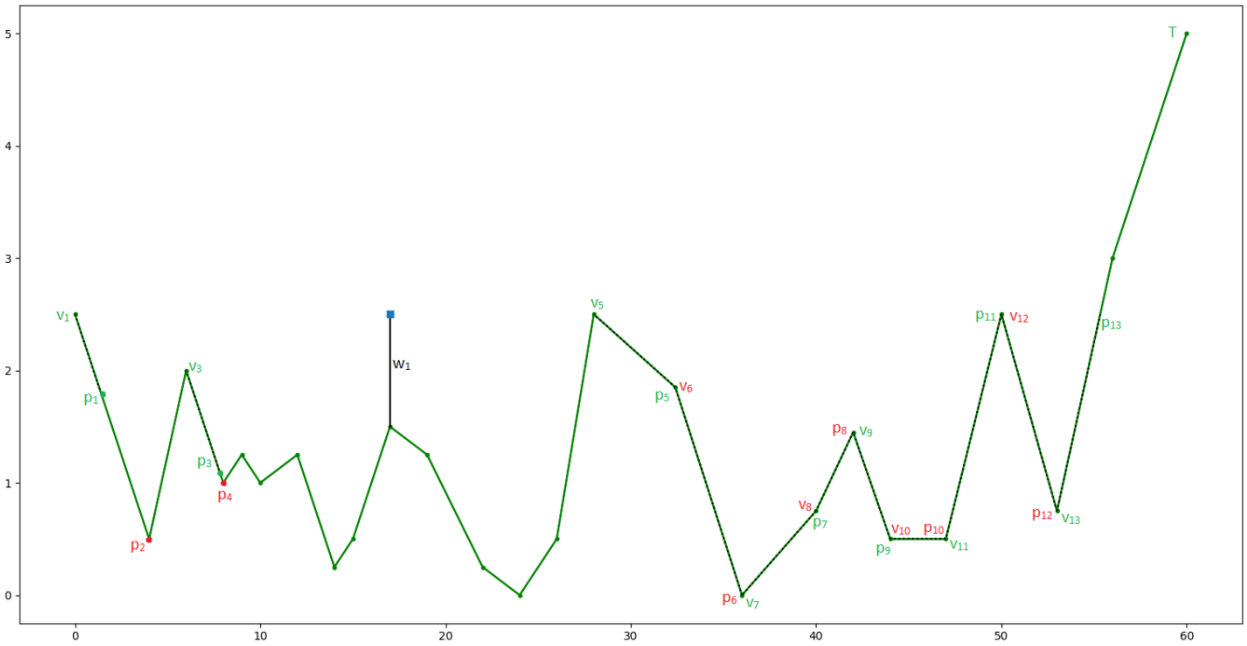
ii) Το p «βλέπει» το v

iii) Το p είναι το αριστερότερο σημείο στο σύνολο P που ικανοποιεί τις i) και ii). Η φυσική σημασία αυτής της συνθήκης είναι ότι το ζεύγος pv είναι το ζεύγος που σχηματίζει ελάχιστη κλίση.

Παρατηρούμε ότι τα «δεξιά» και τα «αριστερά» ζεύγη ορατότητας έχουν μία «καθρεπτική» σχέση. Αυτή η «καθρεπτική» σχέση μας επιτρέπει να πούμε πως τα «δεξιά» ζεύγη στη T για έναν πύργο w_1 είναι τα «αριστερά» ζεύγη ορατότητας μίας άλλης x-μονότονης πολυγωνικής γραμμής T_m (m από το mirror) και ενός πύργου w_{m1}. Οι κορυφές $v_m \epsilon V_m$ της T_m συνδέονται με της κορυφές vεV, εάν θεωρήσουμε ένα δείκτη $0 \leq i \leq n+1$ ώστε $v_m(i)=(x_m(i),y_m(i))$ να είναι η i-οστή κορυφή της T_m και vi=(xi,yi) η i-οστή κορυφή της T, με τις εξής σχέσεις:

$$x_m(i) = x(n-i) + x(n) - x(0) \text{ και } y_m(i) = y(n-i)$$

Τα σημεία $p_m \epsilon P_m$ και τον πύργο w_{m1} τα υπολογίζουμε με όμοιο τρόπο. Έτσι, έχουμε τα p_m και v_m σημεία και τώρα μπορούμε με τη παραπάνω μέθοδο να υπολογίσουμε το σύνολο $\Pi_m(V_m,P_m)$, δηλαδή τα «αριστερά» ζεύγη για την T_m και τον w_{m1}. Για να εξάγουμε τα «δεξιά» ζεύγη ορατότητας αρκεί να «καθρεπτίσουμε» το σύνολο $\Pi_m(V_m,P_m)$.



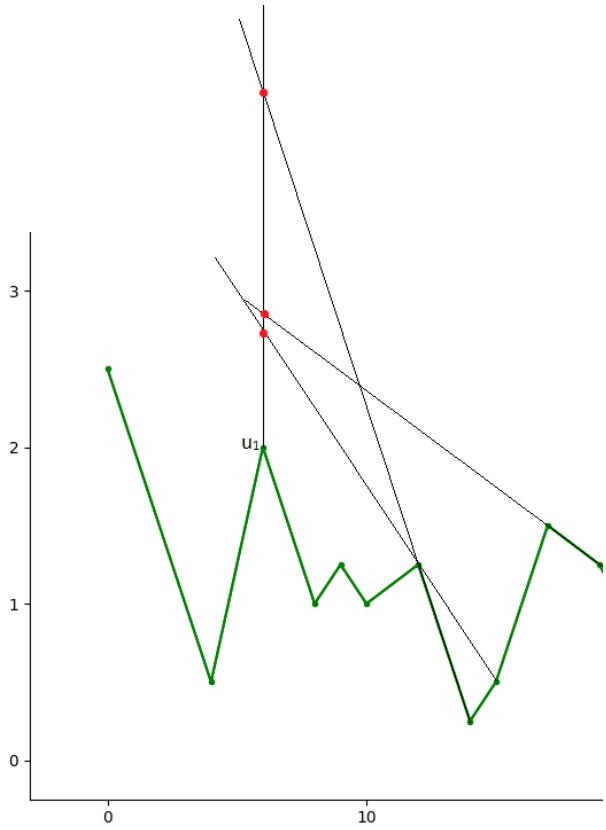
Σχήμα 2.6 *Τα δεξιά ζεύγη ορατότητας που δημιουργούνται από τον πύργο* w_1.

Τα δεξιά ζεύγη ορατότητας τα υπολογίζουμε, όπως τα αριστερά, σε χρόνο $O(n^2 \log n)$.

2.1.4 Κρίσιμα ύψη

Τα κρίσιμα ύψη είναι οι υποψήφιες θέσεις για το $u(h)_1$.

Η επιλογή του h_1 του πρώτου πύργου w_1 δεν γίνεται τυχαία. Ορίζουμε ως κρίσιμα ύψη, για μία κορυφή $u_1 \epsilon V$, τα σημεία τομής, που βρίσκονται πάνω από τη u_1, του φορέα του w_1 με τις ευθείες που πηγάζουν από κάθε κορυφή $v \epsilon V$ και «βλέπουν» στο κατώτερο δυνατό σημείο το φορέα του w_1. Δηλαδή, εάν επιλέξουμε κατάλληλο h_1 ώστε το άνω άκρο του πύργου w_1, $u(h)_1$, να ταυτιστεί με κάποιο από τα κρίσιμα ύψη, τότε αυτό σημαίνει πως επιτηρεί οριακά κάποια κορυφή της T και αν ήταν ελάχιστα κοντύτερο δεν θα «έβλεπε» αυτή την κορυφή.



Σχήμα 2.7 *Τα κρίσιμα ύψη της u_1 που δημιουργούνται από τις μη ορατές κορυφές από την u_1.*

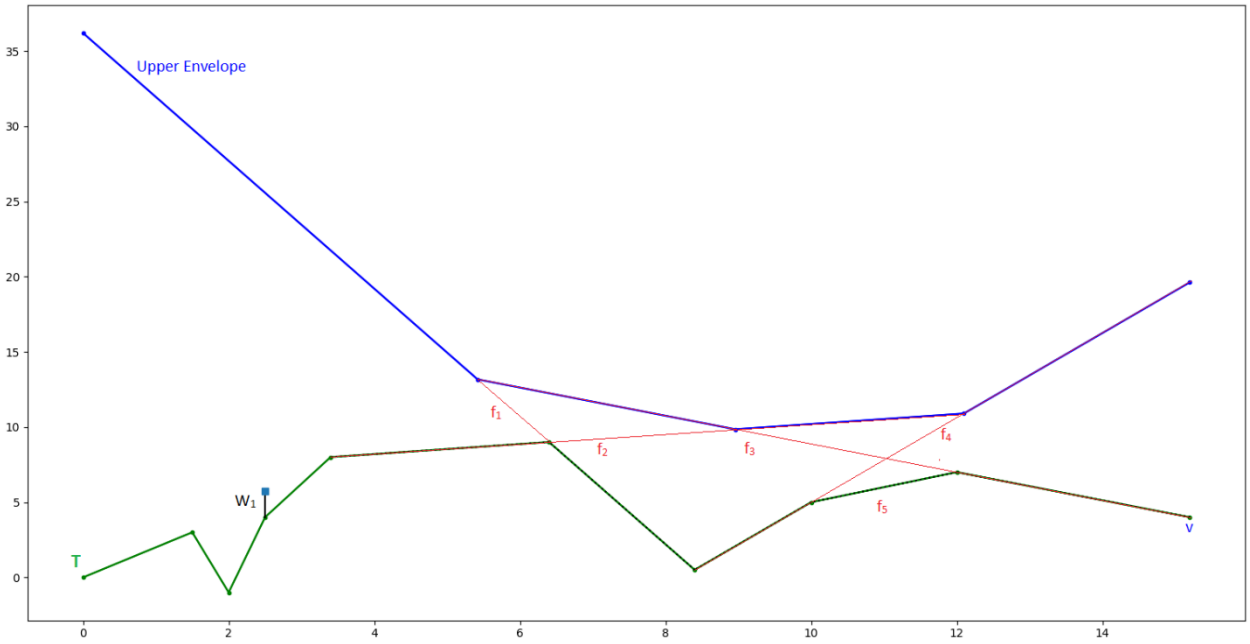
Τα κρίσιμα ύψη για μία κορυφή $v \epsilon V$ τα υπολογίζουμε σε χρόνο O(n).

2.1.5 Upper Envelope

Το upper envelope είναι μία x-μονότονη πολυγωνική γραμμή που ορίζει το κατώτερο σημείο που μπορεί να υψωθεί ο συμπληρωματικός πύργος στην ημι-συνεχή εκδοχή του προβλήματος, ώστε να καλύπτεται η ορατότητα της T.

Στην ημι-συνεχή εκδοχή του προβλήματος, έχουμε πει, πως η βάση του δεύτερου πύργου u_2 μπορεί να βρίσκεται σε οποιοδήποτε σημείο της T. Για να εντοπίσουμε αυτό το σημείο κατασκευάζουμε το Upper Envelope που είναι μία x-μονότονη πολυγωνική γραμμή της οποίας οι ακμές βρίσκονται πάνω από την T ή ταυτίζονται με τμήμα της T. Με δεδομένα, τον υποψήφιο πύργο w_1, τα αριστερά (θέτουμε ως L) και δεξιά (θέτουμε ως R) ζεύγη ορατότητας και τις μη ορατές ακμές (θέτουμε ως E') και μία κορυφή της T vϵV μπορούμε να κατασκευάσουμε το Upper Envelope για τον πύργο w_1 με τον παρακάτω τρόπο:

Υπολογίζουμε τις κλίσεις των φορέων των ευθύγραμμων τμημάτων των L αριστερά της v, R και E' και ταξινομούμε τους φορείς κατά αύξουσα κλίση στο σύνολο F. Επειδή δεν θέλουμε να κάνουμε υπολογισμούς που δεν έχουν νόημα, από τους φορείς με την ίδια κλίση (παράλληλους) κρατάμε μόνο τον χαμηλότερο αυτών. Αρχικά, αν ο πληθάριθμος των φορέων του συνόλου F είναι μεγαλύτερος του ενός τότε βρίσκουμε την τομή των δύο πρώτων φορέων του συνόλου F και κρατάμε την τομή σε ένα σύνολο UE και τους φορείς με την ίδια σειρά σε ένα σύνολο UEF. Για κάθε επόμενο φορέα next_fϵF υπολογίζουμε την τομή του (temp) με τον τελευταίο φορέα last_f που εισαγάγαμε στο UEF. Αν η temp βρίσκεται αριστερά του τελευταίου σημείου last_s που εισαγάγαμε στο UE τότε εξάγουμε το last_s από το UE και το last_f από το UEF και υπολογίζουμε μία νέα τομή temp του φορέα next_f με το νέο τελευταίο φορέα last_fϵUEF. Εάν δεν υπάρχει κάποιο σημείο στο σύνολο UEF ή η τομή temp βρίσκεται δεξιά του last_s τότε απλά προσαρτούμε τη temp στο UE και το next_f στο UEF. Ο αλγόριθμος ολοκληρώνεται όταν ελέγξουμε όλους τους φορείς του F. Ως τελικό Upper Envelope θεωρούμε την ένωση του πρώτου φορέα του UEF έως το πρώτο σημείο της UE με τη x-μονότονη πολυγωνική γραμμή που ορίζουν τα σημεία του συνόλου UE και τον τελευταίο φορέα του EUF που ξεκινάει από το τελευταίο σημείο του UE.



Σχήμα 2.8 Το Upper envelope που υπολογίζουμε από τους φορείς f_1, f_2, f_3, f_4, f_5 που προκύπτουν από τον πύργο w_1 και τη κορυφή V.

Το Upper Envelope το υπολογίζουμε σε χρόνο O(nlogn) λόγω της ταξινόμησης των κλίσεων.

2.1.6 Υπολογισμός συμπληρωματικού πύργου w_2

Αφού έχουμε υπολογίσει τα κρίσιμα ύψη C_h, είμαστε σε θέση να υψώσουμε έναν πύργο w_1 με $u(h)_1 \epsilon C_h$ και στη συνέχεια να υπολογίσουμε τα ζεύγη ορατότητας. Με βάση τα ζεύγη ορατότητας μπορούμε να υπολογίσουμε σε ένα σημείο u_2 τον κοντύτερο συμπληρωματικό δεύτερο πύργο w_2, που μαζί με τον w_1 καλύπτουν την ορατότητα της T με βάση το u_2 για το οποίο στη διακριτή εκδοχή του προβλήματος ισχύει $u_2 \epsilon V$, ενώ στην ημι-συνεχή το $u_2 \epsilon T$, ύψος h_2 και άνω άκρο $u(h)_2$ ως εξής:

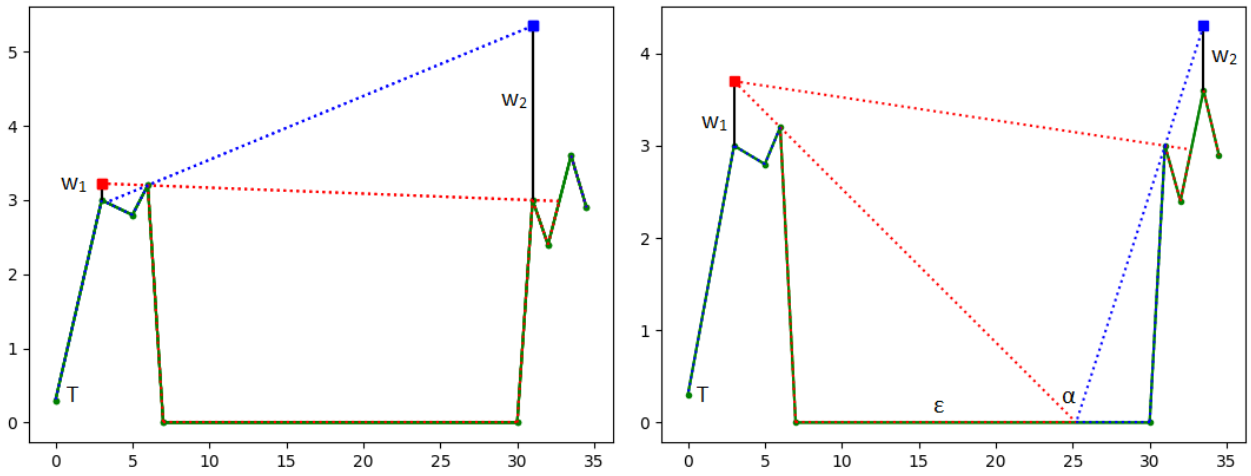
Στη διακριτή εκδοχή, το $u(h)_2$ ταυτίζεται με την υψηλότερη τομή των φορέων των μη ορατών ακμών από τον w_1 και των ευθύγραμμων τμημάτων που σχηματίζουν τα αριστερά ζεύγη $(p,v) \epsilon \Pi(P,V)$ που τα p,v ταυτίζονται ή βρίσκονται αριστερά της u_2 με τον κατακόρυφο φορέα που διέρχεται από το u_2, με $u_2 \epsilon V$ μία κορυφή δεξιά του u_1. Αναζητούμε τη βάση $u_2 \epsilon T$ του συμπληρωματικού πύργου w_2 δεξιά του u_1, διότι θέτουμε πρώτη υποψήφια βάση όλες τις κορυφές από αριστερά προς τα δεξιά εξασφαλίζοντας έτσι τον υπολογισμό των βέλτιστων ζευγών φύλαξης των οποίων οι βάσεις έχουν μικρότερες ή ίσες τετμημένες της u_1.

Στη ημι-συνεχή εκδοχή, το $u(h)_2$ το υπολογίζουμε με τη βοήθεια του Upper Envelope που το κατασκευάζουμε βασισμένο σε κάποια ακμή e της T. Πιο συγκεκριμένα, όπως αναφέραμε και παραπάνω, χρησιμοποιούμε τα αριστερά ζεύγη ορατότητας που βρίσκονται αριστερά της κορυφής/πιθανής βάσης του u_2, επειδή κάποιος φορέας δεξιά αυτής της κορυφής μπορεί να παραπλανήσει την λύση με πρόσθετη πληροφορία. Έτσι για κάθε ακμή e στην T, κατασκευάζουμε το Upper Envelope από τους φορείς των E', R και τους φορείς του L αριστερά της δεξιότερης κορυφής της ακμής. Τα σημεία που εξετάζουμε ως πιθανές βάσεις u_2 είναι η δεξιά κορυφή της ακμής και οι τομές των κατακόρυφων ημι-ευθειών που ξεκινούν από τις κορυφές του Upper Envelope με την ακμή e. Για κάθε πιθανή βάση υπολογίζουμε την κατακόρυφη απόσταση από το Upper Envelope και διατηρούμε αυτή με την μικρότερη.

2.1.7 Βελτιστοποίηση

Η λύση μπορεί να μην είναι ένα κρίσιμο ύψος αλλά λίγο πιο κάτω ή πάνω από αυτό.

Τα κρίσιμα ύψη και κατ' επέκταση η λύση του προβλήματος βασίζονται στην οριακή ορατότητα κάποιων κορυφών. Η λύση όμως είναι πολύ πιθανό να είναι αποτέλεσμα οριακής ορατότητας ενός σημείου α μίας ακμής ε με την έννοια ότι ο ένας πύργος επιτηρεί την αριστερή κορυφή έως το σημείο α της ακμής ε και ο άλλος πύργος επιτηρεί την ε από το σημείο α έως την δεξιά κορυφή (Σχήμα 2.9).



Σχήμα 2.9 *Η καλύτερη λύση εάν λάβουμε υπόψιν μόνο τα κρίσιμα ύψη(αριστερά). Βελτιστοποιημένη λύση εντοπίζοντας το σημείο α της ε (δεξιά)*

Εμείς υλοποιήσαμε έναν απλό αλγόριθμο βελτιστοποίησης, όπου βρίσκουμε την διαφορά ύψους του ζεύγους w_1,w_2 και τη διαιρούμε σε πολλά σημεία (j). Εφαρμόζοντας εκ νέου τον αλγόριθμο εύρεσης του δεύτερου πύργου θεωρώντας ως κρίσιμα ύψη τα j βρίσκουμε καλύτερη ή ίδια λύση για σχετικά μικρό υπολογιστικό κόστος.

2.2 Ο Αλγόριθμος

Ο αλγόριθμος χρησιμοποιεί παραμετρική αναζήτηση και σε μία πιο απλή εκδοχή ακολουθεί τα παρακάτω βήματα:

2.2.1 Διακριτή εκδοχή του προβλήματος

Δεδομένα: Το σύνολο V

Έξοδος: Τις βάσεις u_1,u_2ϵV και τα ύψη h_1, h_2

1) Αρχικοποιούμε τις βάσεις u_1,u_2 και τα ύψη h_1, h_2.
2) Υπολογίζουμε τις ακμές της T σύμφωνα με τις κορυφές V και τις αποθηκεύουμε στο σύνολο E.
3) Εκκίνηση επαναληπτικής διαδικασίας για κάθε κορυφή viϵV.
    i) Υπολογίζουμε τα κρίσιμα ύψη και τα προσαρτούμε σε ένα σύνολο C_h κατά αύξουσα σειρά τεταγμένης.
    ii) Προετοιμάζουμε δυαδική αναζήτηση θέτοντας left το δείκτη του πρώτου κρίσιμου ύψους της C_h και right το δείκτη του τελευταίου και αρχικοποιώντας τις μεταβλητές του δεύτερου πιθανού πύργου.
    iii) Εκκίνηση επαναληπτικής διαδικασίας όσο δεν βρέθηκε ελάχιστο κοινό ύψος των πύργων και το right≥left.
        a) Θέτουμε ως πιθανή πρώτη βάση cur_u_1 τη vi και ως πιθανό πρώτο ύψος cur_h_1 το μεσαίο στοιχείο από το σύνολο C_h από το left μέχρι το right.
        b) Υπολογίζουμε το σύνολο P που αποτελείται από τα δεξιά άκρα των ολικώς ή μερικώς μη ορατών ακμών που βρίσκονται δεξιά της vi.
        c) Υπολογίζουμε τα αριστερά ζεύγη ορατότητας για τα σημεία P στο σύνολο L.
        d) Αρχικοποιούμε ένα κενό σύνολο B, το οποίο θα ενημερώνουμε με τα αριστερά ζεύγη από το σύνολο L αναλόγως της κορυφή cur_u_2 που θα εξετάζουμε.

e) Για όλες τις κορυφές cur_u_2ϵV δεξιά της cur_u_1.
   - Υπολογίζουμε το υψηλότερο σημείο τομής temp του φορέα της κατακόρυφης ημι-ευθείας που πηγάζει από την cur_u_2 και έχει φορά προς τα πάνω, με τους φορείς της ένωσης E'UB.
   - Η διαφορά της τεταγμένης του cur_u_2 από την τεταγμένη της temp είναι το τωρινό ύψος του δεύτερου υποψήφιου πύργου το οποίο συγκρίνουμε με το έως τώρα χαμηλότερο υποψήφιο ύψος πύργου έτσι ώστε να κρατήσουμε τις μεταβλητές του κοντύτερου.
   - Πριν γίνει ο έλεγχος της επόμενης δεξιότερης κορυφής ως υποψήφια βάση δεύτερου πύργου ενημερώνουμε με τα κατάλληλα αριστερά ζεύγη ορατότητας του L το σύνολο B.

f) Έχοντας υπολογίσει τον κοντύτερο δεύτερο πύργο που μπορούμε να υψώσουμε δεξιά του πρώτου υποψηφίου, ελέγχουμε τα ύψη cur_h_1 και cur_h_2:
   - Εάν είναι ίσα τότε βρήκαμε το ελάχιστο κοινό ύψος για την vi κορυφή ως βάση πρώτου πύργου.
   - Εάν ο πρώτος πύργος είναι μικρότερος, (τότε υποψιαζόμαστε πως αν αυξήσουμε σε κάποιο ψηλότερο ύψος από τα C_h τον πρώτο πύργο ο δεύτερος που θα προκύψει να χαμηλώσει λίγο) θέτουμε ως left το μεσαίο δείκτη ανάμεσα των τωρινών left και right αυξημένο κατά ένα. Αναλόγως εάν ο πρώτος είναι μεγαλύτερος, θέτουμε ως right το μεσαίο δείκτη ανάμεσα των τωρινών left και right μειωμένο κατά ένα.

iv) Ελέγχουμε εάν μπορεί να βελτιστοποιηθεί η λύση για το w_1.

v) Ελέγχουμε αν αυτό ή το προηγούμενο υποψήφιο ζεύγος κάλυψης ορατότητας της T έχουν το μικρότερο ελάχιστο κοινό ύψος και προφανώς διατηρούμε αυτό με το μικρότερο.

4) Ορίζουμε ως κοινό ύψος των πύργων το μεγαλύτερο ύψος.
5) Επιστρέφουμε τις βάσεις των πύργων και το ύψος τους.

2.2.2 Ημι-συνεχής εκδοχή του προβλήματος

Δεδομένα: Το σύνολο V

Έξοδος: Τη βάση u_1ϵV, τις συντεταγμένες της βάσης u_2 και τα ύψη h_1, h_2

1) Αρχικοποιούμε τις βάσεις u_1,u_2 και τα ύψη h_1, h_2.

2) Υπολογίζουμε τις ακμές της T σύμφωνα με τις κορυφές V και τις αποθηκεύουμε στο σύνολο E.
3) Εκκίνηση επαναληπτικής διαδικασίας για κάθε κορυφή viϵV.
   i) Υπολογίζουμε τα κρίσιμα ύψη και τα προσαρτούμε σε ένα σύνολο C_h κατά αύξουσα σειρά τεταγμένης.
   ii) Προετοιμάζουμε δυαδική αναζήτηση θέτοντας left το δείκτη του πρώτου κρίσιμου ύψους της C_h και right το δείκτη του τελευταίου και αρχικοποιώντας τις μεταβλητές του δεύτερου πιθανού πύργου.
   iii) Εκκίνηση επαναληπτικής διαδικασίας όσο δεν βρέθηκε ελάχιστο κοινό ύψος των πύργων και το right≥left.
      a) Θέτουμε ως πιθανή πρώτη βάση cur_u_1 τη vi και ως πιθανό πρώτο ύψος cur_h_1 το μεσαίο στοιχείο από το σύνολο C_h από το left μέχρι το right.
      b) Υπολογίζουμε τα δεξιά άκρα των ολικώς ή μερικώς μη ορατών ακμών που βρίσκονται δεξιά της vi και τα βάζουμε στο σύνολο P.
      c) Υπολογίζουμε τα αριστερά άκρα των ολικώς ή μερικώς μη ορατών ακμών που βρίσκονται αριστερά της vi και τα βάζουμε στο σύνολο P.
      d) Υπολογίζουμε τα αριστερά και δεξιά ζεύγη ορατότητας για όλα τα σημεία P στα σύνολα L και R αντίστοιχα διατεταγμένα κατά αύξουσα τετμημένη της κορυφής του ζεύγους.
      e) Αρχικοποιούμε κενό σύνολο B το οποίο θα ενημερώνουμε με τα αριστερά ζεύγη αναλόγως την ακμή που θα εξετάζουμε.
      f) Υπολογίζουμε το πρώτο ζεύγος πύργων με πρώτο πύργο στην αριστερότερη κορυφή της T.
      g) Αφαιρούμε από το σύνολο R όλα τα δεξιά ζεύγη που σχετίζονται με την αριστερότερη κορυφή της T.
      h) Εκκίνηση επαναληπτικής διαδικασίας για κάθε ακμή της T από αριστερά προς τα δεξιά:
         - Υπολογίζουμε το Upper Envelope UE που προκύπτει από τους φορείς της ένωσης των συνόλων E’UBUR.
         - Υπολογίζουμε τις κορυφές του UE που οι τετμημένες τους είναι εντός των τετμημένων των άκρων της ακμής που εξετάζουμε και ελέγχουμε αν η απόσταση των κορυφών από την ακμή είναι πιο μικρή από τον προηγούμενο μικρότερο πύργο που είχαμε υπολογίσει και σε αυτή την περίπτωση τον αντικαθιστούμε.
         - Ενημερώνουμε το B με τα αριστερά ζεύγη που σχετίζονται με τη δεξιά κορυφή της ακμής.

- Υπολογίζουμε εάν η δεξιά κορυφή της ακμής που εξετάζουμε δημιουργεί ζεύγος ορατότητας με τωρινό ελάχιστο κοινό ύψος βάσει των φορέων των συνόλων E'UBUR.

i) Έχοντας υπολογίσει τον κοντύτερο δεύτερο πύργο που μπορούμε να υψώσουμε αριστερά ή δεξιά του πρώτου υποψηφίου, ελέγχουμε τα ύψη cur_h_1 και cur_h_2:

- Εάν είναι ίσα τότε βρήκαμε το ελάχιστο κοινό ύψος για την vi κορυφή ως βάση πρώτου πύργου.
- Εάν ο πρώτος πύργος είναι μικρότερος, (τότε υποψιαζόμαστε πως αν αυξήσουμε σε κάποιο ψηλότερο ύψος από τα C_h τον πρώτο πύργο ο δεύτερος που θα προκύψει να χαμηλώσει λίγο) θέτουμε ως left το μεσαίο δείκτη ανάμεσα των τωρινών left και right αυξημένο κατά ένα. Αναλόγως εάν ο πρώτος είναι μεγαλύτερος, θέτουμε ως right το μεσαίο δείκτη ανάμεσα των τωρινών left και right μειωμένο κατά ένα.

iv) Ελέγχουμε εάν μπορεί να βελτιστοποιηθεί η λύση για το w_1.

v) Ελέγχουμε αν αυτό ή το προηγούμενο υποψήφιο ζεύγος κάλυψης ορατότητας της T έχουν το μικρότερο ελάχιστο κοινό ύψος και προφανώς διατηρούμε αυτό με το μικρότερο.

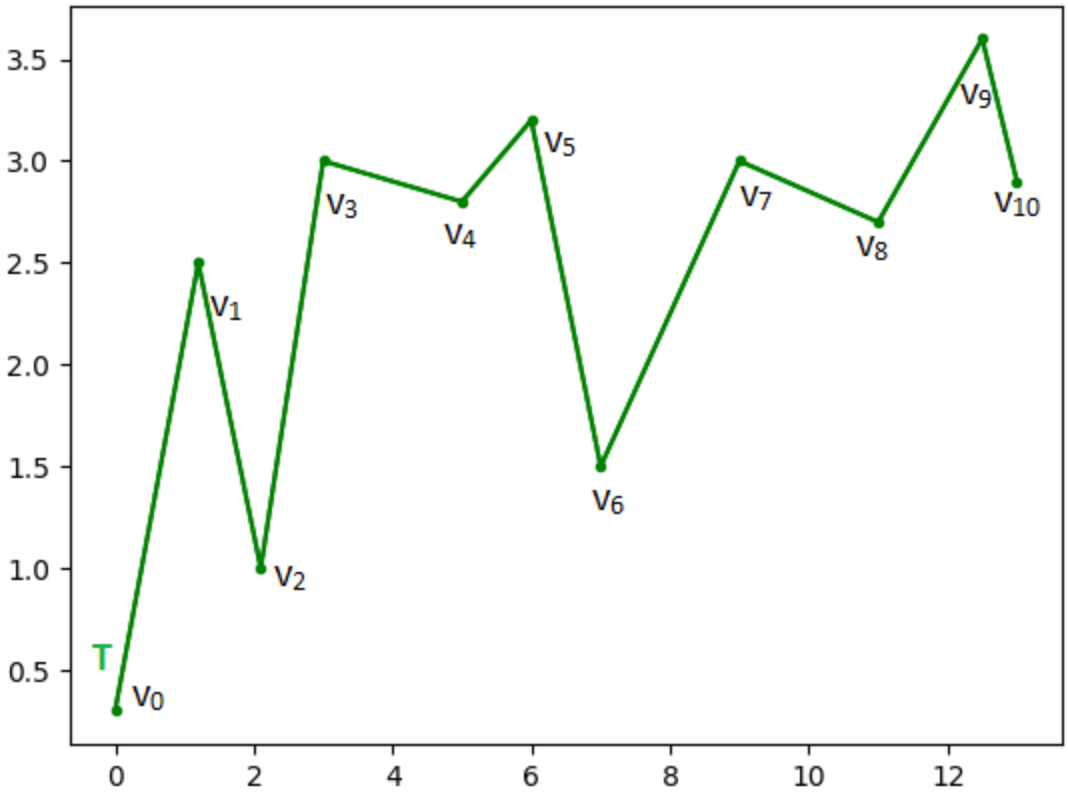
4) Ορίζουμε ως κοινό ύψος των πύργων το μεγαλύτερο ύψος.
5) Επιστρέφουμε τις βάσεις των πύργων και το ύψος τους.

2.3 Παραδείγματα Εφαρμογής του Αλγορίθμου

Παρακάτω αναλύουμε δύο παραδείγματα εφαρμογής του αλγορίθμου. Ένα για τη διακριτή εκδοχή του προβλήματος και ένα για την ημι-συνεχή εκδοχή του προβλήματος.

2.3.1 Παράδειγμα Εφαρμογής του Αλγορίθμου της διακριτής εκδοχής

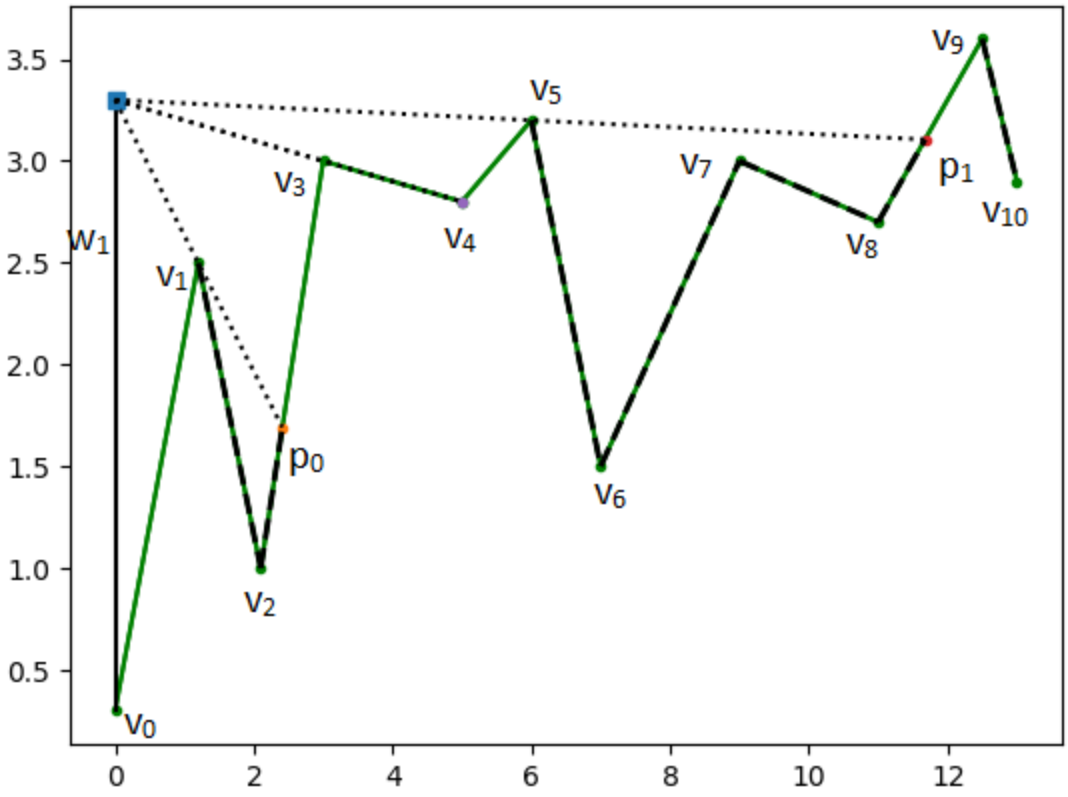
Ας χρησιμοποιήσουμε για το συγκεκριμένο παράδειγμα την x-μονότονη πολυγωνική γραμμή T όπως είναι στο Σχήμα 2.10.



Σχήμα 2.10 *Η x-μονότονη πολυγωνική γραμμή του παραδείγματος.*

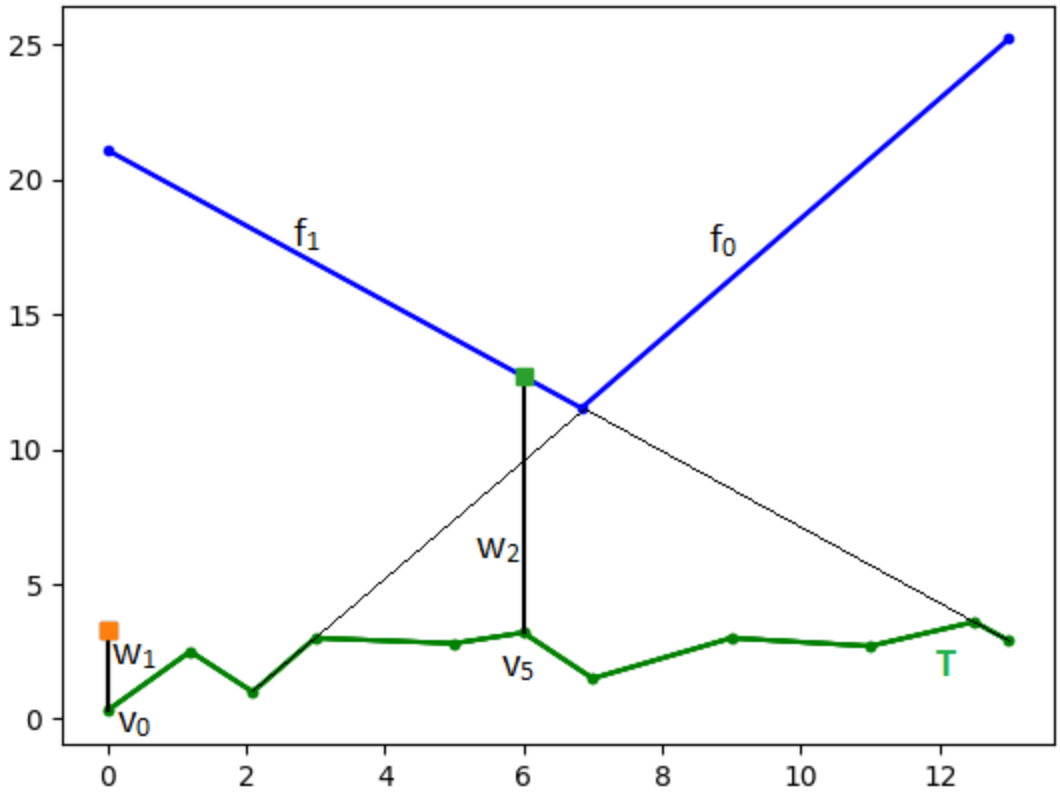
Η T αποτελείται από έντεκα κορυφές: $v_0 = (0, 0.3)$, $v_1 = (1.2, 2.5)$, $v_2 = (2.1, 1)$, $v_3 = (3, 3)$, $v_4 = (5, 2.8)$ $v_5 = (6, 3.2)$, $v_6 = (7, 1.5)$, $v_7 = (9, 3)$, $v_8 = (11, 2.7)$, $v_9 = 12.5, 3.6)$, $v_{10} = (13, 2.9)$.

Ο αλγόριθμος ξεκινά την αναζήτηση για την βάση του πρώτου υποψήφιου πύργου w_1 από την αριστερότερη κορυφή οπότε θέτουμε ως u_1 τη v_0 και υπολογίζουμε τα κρίσιμα ύψη C_h=[0.3, 0.8, 2.17, 2.8, 2.83, 3.3, 3.6, 4.35, 4.5, 13.4, 21.1]. Από τα κρίσιμα ύψη, επιλέγουμε το μεσαίο το οποίο ορίζει τη θέση του άνω άκρου του πύργου. Έτσι έχουμε υπολογίσει τον πρώτο υποψήφιο πύργο του οποίου η βάση είναι η κορυφή v_0 και το ύψος h = 3 το οποίο το υπολογίσαμε από το κρίσιμο ύψος που δημιουργεί η ακμής (v_3,v_4). Στο Σχήμα 2.11 βλέπουμε καλύτερα τον πύργο w_1 καθώς και την ορατότητα που καλύπτει επιτηρώντας οριακά την ακμή (v_3,v_4).



Σχήμα 2.11 *Ο w_1 όπως υπολογίστηκε από την ακμή (v_3,v_4) και η ορατότητα που καλύπτει στη T*

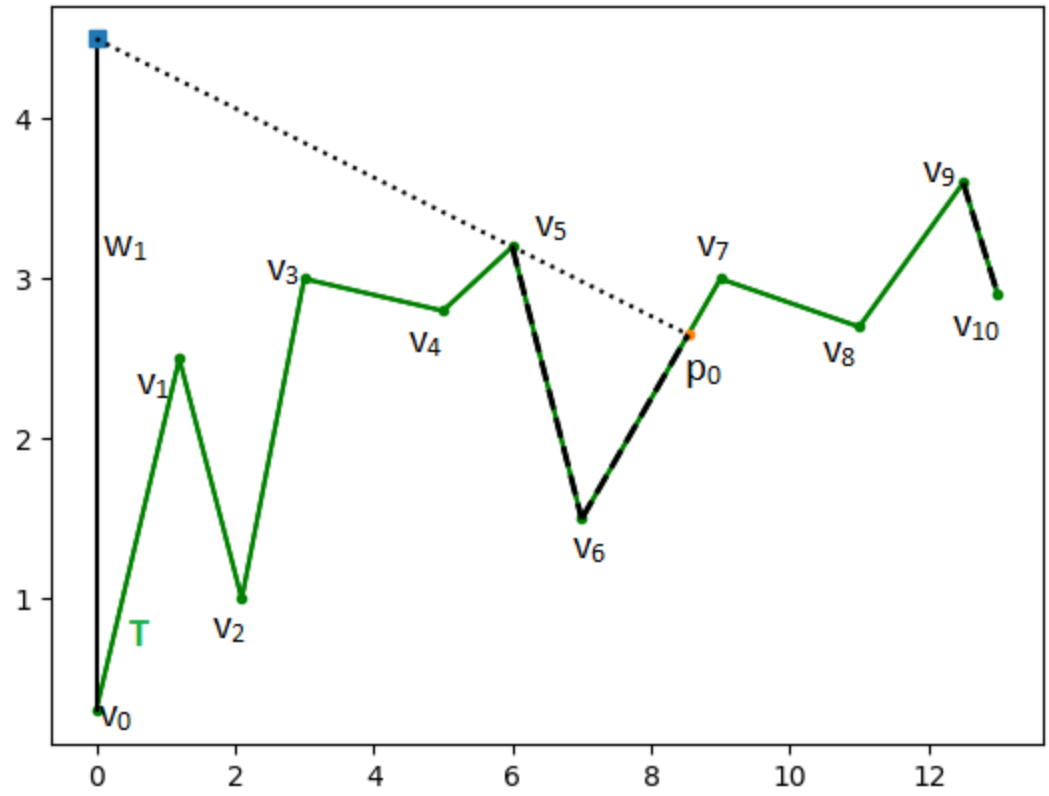
Στη συνέχεια, για κάθε κορυφή δεξιά του w_1 υπολογίζουμε τους απαιτούμενους φορείς από τα αριστερά ζεύγη ορατότητας και από τις μη ορατές ακμές και υπολογίζουμε τον συμπληρωματικό πύργο w_2, που καλύπτει τα σκοτεινά μέρη της T που δημιούργησε ο w_1, με το μικρότερο ύψος. Στη συγκεκριμένη περίπτωση, οι φορείς των αριστερών ζευγών ορατότητας περιέχονται στους φορείς των ολικώς ή μερικώς μη ορατών ακμών, οπότε δεν αλλάζουν οι φορείς που χρησιμοποιούμε σε όλη την αναζήτηση του συμπληρωματικού πύργου w_2 όπως φαίνεται στο Σχήμα 2.12.



Σχήμα 2.12 *Ο υπολογισμός του w_2 με το χαμηλότερο ύψος όπως ορίζεται από τους φορείς f_0, f_1 οι οποίοι υπερκαλύπτουν τους υπόλοιπους φορείς.*

Για το w_1 με βάση την κορυφή v_0 και ύψος h_1=3, υπολογίσαμε πως ο χαμηλότερος πύργος με τον οποίο μπορεί να επιτηρηθεί η T είναι ο πύργος w_2 με βάση την κορυφή v_5 και ύψος h_2=9.5. Τα αποτελέσματα αυτά τα διατηρούμε ως η καλύτερη έως τώρα λύση για πρώτο πύργο με βάση τη v_0 με κοινό ύψος h=9.5.

Επειδή το ύψος του πρώτου πύργου είναι μεγαλύτερο από το ύψος του δεύτερου συνεχίζουμε τη δυαδική αναζήτηση στα κρίσιμα ύψη επιλέγοντας το μεσαίο κρίσιμο ύψος από τα ψηλότερα από αυτό που επιλέξαμε αρχικά C_h=[3.6, 4.35, 4.5, 13.4, 21.1]. Η επόμενη επιλογή είναι το h=4.2 που δημιουργήθηκε από την ακμή (v_1, v_2). Ο νέος πύργος w_1 και η ορατότητα που καλύπτει φαίνονται στο Σχήμα 2.13.



Σχήμα 2.13 *Η ορατότητα του νέου πύργου w_1*

Αφού υπολογιστούν οι φορείς των αριστερών ζευγών ορατότητας και των μερικώς ή ολικώς μη ορατών ακμών υπολογίζεται για κάθε κορυφή ο πύργος με το μικρότερο ύψος h_2 που καλύπτει την ορατότητα και σε αυτή την περίπτωση είναι ο πύργος με βάση την κορυφή v_9 και ύψος h_2=2.025 όπως ορίζεται από τους φορείς των ευθύγραμμων τμημάτων (v_6,p_0) και (v_9,v_{10}).

Αφού συγκρίνουμε αυτή τη λύση με την προηγούμενη που έχουμε διατηρήσει, συμπεραίνουμε ότι το h=4.2 είναι μικρότερο από το h=9.5. Έτσι θέτουμε ως καλύτερη τωρινή λύση με πρώτο πύργο w_1 με βάση την κορυφή v_0 και ύψος h_1=4.2 και τον πύργο w_2 με βάση την κορυφή v_9 και ύψος το h_2=2.025.

Επιστροφή στα κρίσιμα ύψη και επιλογή του μεσαίου μικρότερου από αυτά που έμειναν στη λίστα C_h=[3.6, 4.35], δηλαδή h_1=3.3. Ακολουθώντας με την ίδια διαδικασία υπολογίζουμε το δεύτερο πύργο w_2 με βάση τη v_5 και ύψος h_2=9.5. Προφανώς χειρότερη λύση από το h=4.2 και για αυτό δεν τη διατηρούμε.

Ο δεύτερος πύργος που υπολογίσαμε είναι μεγαλύτερος από τον πρώτο. Έτσι, εξετάζουμε από τα κρίσιμα ύψη το μεγαλύτερο και μεσαίο από την λίστα Ch=[4.35], δηλαδή

θέτουμε h_1=4.05. Το αποτέλεσμα για τον w_2 είναι το ίδιο (βάση τη v_5 και ύψος h_2=9.5) και επομένως, η καλύτερη λύση μετά τον έλεγχο για λύση με πρώτο πύργο με βάση στην κορυφή v_0 είναι: u_1=v_0, h_1=4.2 και u_2=v_9, h_2=2.025.

Εκτελώντας την ίδια διαδικασία για την δεύτερη κορυφή v_1=(1.2, 2.5), υπολογίζουμε αρχικά τα κρίσιμα ύψη C_h=[2.88, 2.9, 3.18, 3.52, 4.17, 11.36, 19.42]. Πρώτος πύργος που εξετάζουμε για την κορυφή v_1 είναι ο w_1 με u_1=v_1 και h_1=1.02. Το αποτέλεσμα του υπολογισμού του δεύτερου πύργου είναι ο w_2 με u_2=v_9 και h_1=2.025.

Συνεχίζοντας τον αλγόριθμο, εξετάζοντας και τα άλλα κρίσιμα ύψη με τον ίδιο τρόπο, τα ζεύγη που προκύπτουν (u_1=v_1 με h_1=8.86, u_2=v_9 με h_2=0.0 και u_1=v_1 με h_1=1.67, u_2=v_9 με h_2=2.025) δεν αλλάζουν το πρώτο αποτέλεσμα και συνεπώς το καλύτερο ζεύγος πύργων που επιτηρούν τη T για κορυφή του πρώτου πύργου τη v_1 είναι ο w_1 με u_1=v_1 και h_1=1.02 και ο w_2 με u_2=v_9 και h_1=2.025. Ελέγχοντας αυτή τη λύση με την καλύτερη μέχρι τώρα παρατηρούμε ότι το ελάχιστο κοινό ύψος που υπολογίσαμε για αυτή την κορυφή είναι μικρότερο από την τρέχουσα λύση. Άρα θέτουμε ως τρέχουσα λύση το ζεύγος w_1 με u_1=v_1 και h_1=1.02 και w_2 με u_2=v_9 και h_1=2.025.

Ακολουθώντας την ίδια λογική υπολογίζουμε τα ζεύγη και για τις επόμενες κορυφές της T, χωρίς όμως να υπολογίσουμε καλύτερη λύση από το h=2.025. Πιο συγκεκριμένα, τα χαμηλότερα ζευγάρια πύργων των επόμενων κορυφών ως βάσεις του πρώτου είναι:

w_1 με u_1=v_2 και h_1=3.15 και w_2 με u_2=v_9 και h_2=2.025

w_1 με u_1=v_3 και h_1=2.8 και w_2 με u_2=v_9 και h_2=2.025

w_1 με u_1=v_4 και h_1=6.667 και w_2 με u_2=v_9 και h_2=0.0

w_1 με u_1=v_5 και h_1=8.1 και w_2 με u_2=v_9 και h_2=0.0

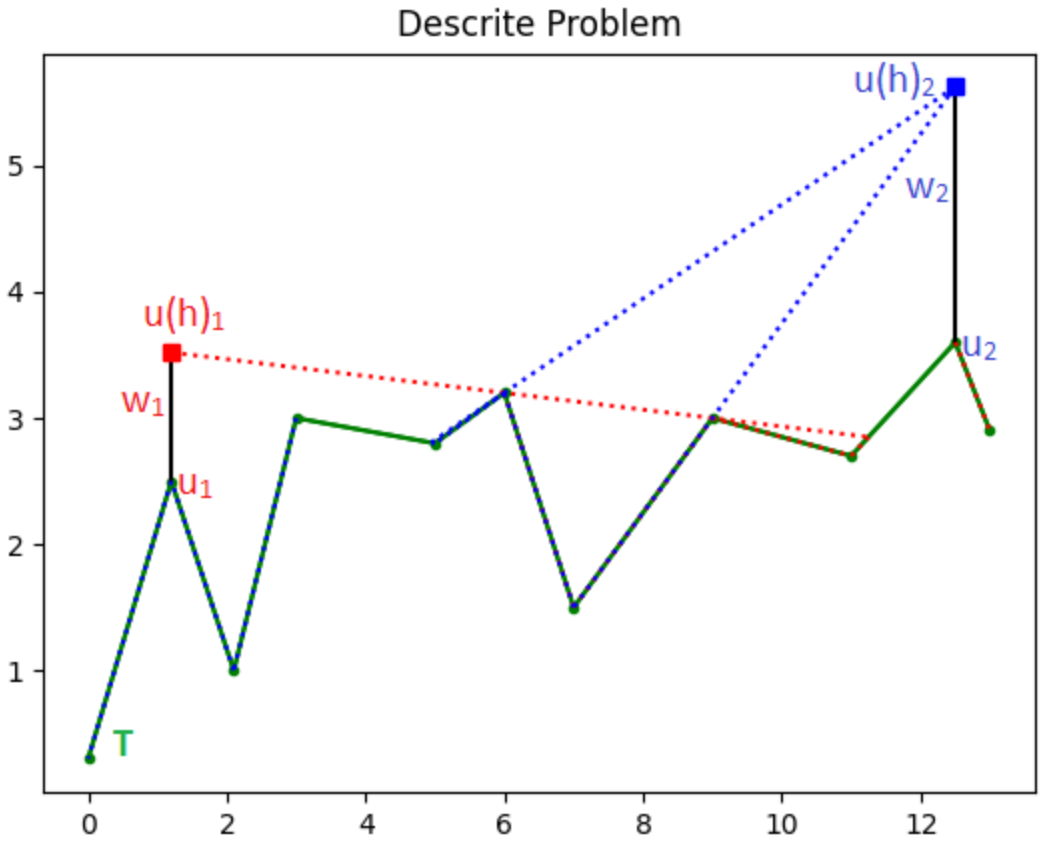
w_1 με u_1=v_6 και h_1=11.63 (βλέπει όλη την T) και w_2 με u_2=v_7 και h_2=0.0

w_1 με u_1=v_7 και h_1=13.8 (βλέπει όλη την T) και w_2 με u_2=v_8 και h_2=0.0

w_1 με u_1=v_8 και h_1=18.08 (βλέπει όλη την T) και w_2 με u_2=v_9 και h_2=0.0

w_1 με u_1=v_9 και h_1=20.51 (βλέπει όλη την T) και w_2 με u_2=v_{10} και h_2=0.0

Έτσι, όπως φαίνεται και στο Σχήμα 2.14, η λύση είναι το ζεύγος πύργων w_1 με u_1=v_1 και h_1=1.02 και w_2 με u_2=v_9 και h_1=2.025. Άρα το ελάχιστο κοινό ύψος h που μπορεί να καλύψει την ορατότητα της T είναι h=2.025.



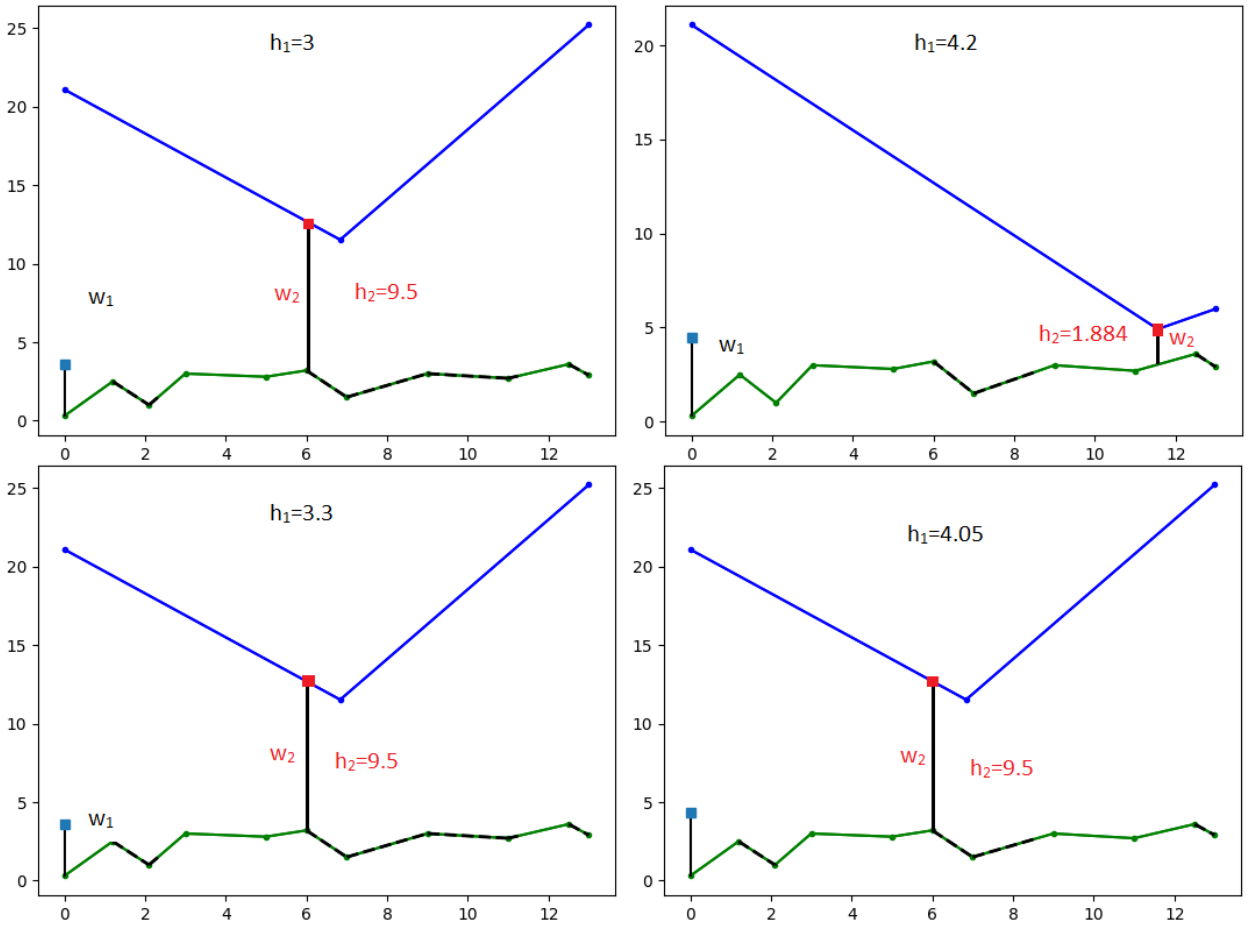
Σχήμα 2.14 *Η T και οι δύο πύργοι με βάσεις που την επιτηρούν πλήρως με ελάχιστο κοινό ύψος το h_2=2.025.*

2.3.2 Παράδειγμα εφαρμογής της ημι-συνεχούς εκδοχής του αλγορίθμου

Για την ημι-συνεχή λύση του προβλήματος θα χρησιμοποιήσουμε πάλι την x-μονότονη πολυγωνική γραμμή T του Σχήματος 2.10. Όπως μπορούμε να δούμε στην ενότητα 2.2 Ο Αλγόριθμος οι διαφορές των δύο αλγορίθμων εντοπίζονται στον τρόπο υπολογισμού του δεύτερου πύργου, δηλαδή την κατασκευή του upper envelope. Στη διακριτή εκδοχή, η επιλογή του δεύτερου πύργου γίνεται από επιλεγμένους φορείς από τα αριστερά ζεύγη ορατότητας και όλους τους φορείς των μερικώς ή ολικώς μη ορατών ακμών. Για το upper envelope, χρησιμοποιούμε επιπροσθέτως τους φορείς των δεξιών ζευγών ορατότητας.΄

Τα κρίσιμα ύψη για κάθε κορυφή της T δεν αλλάζουν C_h=[0.3, 0.8, 2.17, 2.8, 2.83, 3.3, 3.6, 4.35, 4.5, 13.4, 21.1], και αυτό σημαίνει πως οι πρώτοι υποψήφιοι πύργοι θα είναι ίδιοι με το παράδειγμα της διακριτής εκδοχής. Ο πρώτος υποψήφιος πύργος w_1 είναι αυτός του Σχήματος 2.11. Ο αλγόριθμος θα κατασκευάσει και θα ελέγξει τις κορυφές του upper envelope που

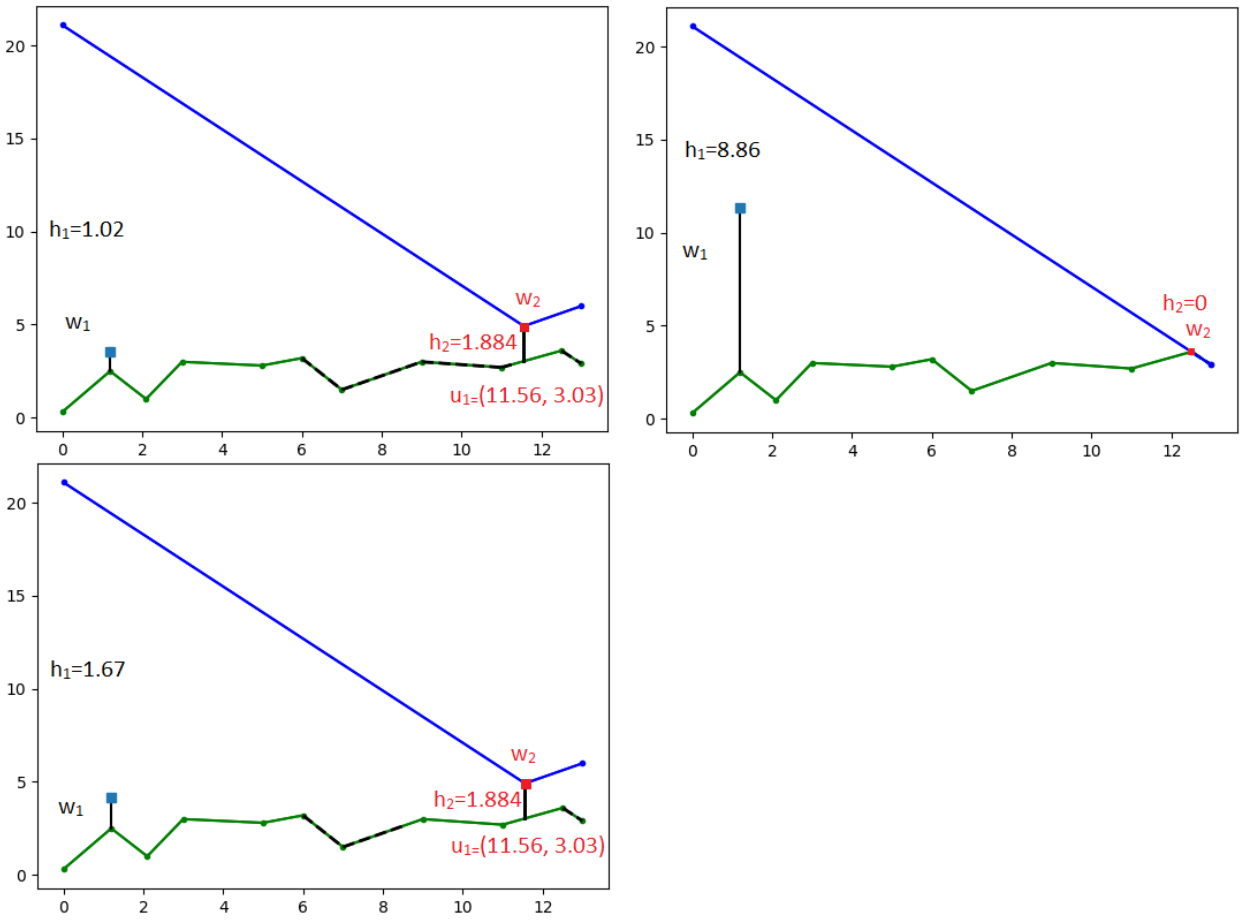
βρίσκονται εντός των ακμών της T για πιθανό $u(h)_2$ και τις κορυφές της T για πιθανή βάση u_2 του δεύτερου πύργου. Για βάση την v_0 και ύψη h_1=3/4.2/3.3/4.05 κατασκευάζουμε τα upper envelope όπως φαίνεται στο Σχήμα 2.15.



Σχήμα 2.15 *Τα upper envelope και ο δεύτερος κοντύτερος πύργος που ορίζουν για τον πρώτο πύργο με βάση τη v_0 και ύψη h_1=3/4.2/3.3/4.05.*

Από το Σχήμα 2.15 φαίνεται καθαρά ότι η προσωρινή λύση είναι το ζεύγος w_1 με βάση τη $u_1=v_0$ και ύψος το h_1=4.2, w_2 με βάση τη u_2=(11.56, 3.03) και ύψος το h_2=1.884. Δηλαδή, ελάχιστο κοινό ύψος που οι δύο πύργοι επιτηρούν την T, h=4.2.

Συνεχίζοντας στη δεύτερη κορυφή ως υποψήφια βάση πρώτου πύργου, έχουμε τα κρίσιμα ύψη C_h=[2.88, 2.9, 3.18, 3.52, 4.17, 11.36, 19.42]. Από αυτά, ο αλγόριθμος θα επιλέξει με τη σειρά τα ύψη h_1=1.02/8.86/1.67 και θα υπολογίσει τα upper envelope όπως φαίνονται στο Σχήμα 2.16.



Σχήμα 2.16 *Τα upper envelope και ο δεύτερος κοντύτερος πύργος που ορίζει ο πρώτος πύργος με βάση τη v_1 και ύψη h_2=1.02/8.86/1.67.*

Όπως καταλαβαίνουμε και από το Σχήμα 2.16 η καλύτερη λύση για πρώτο πύργο με κορυφή τη v_1 είναι το ζεύγος w_1 με βάση τη $u_1=v_1$ και ύψος το h_1=1.02, w_2 με βάση τη u_2=(11.56, 3.03) και ύψος το h_2=1.884. Δηλαδή, ελάχιστο κοινό ύψος που οι δύο πύργοι επιτηρούν την T, h=1.884.

Αν εξαντλήσουμε τον αλγόριθμο θα συμπεράνουμε ότι το h=1.884 είναι όντως η βέλτιστη λύση με πρώτη βάση $u_1=v_1$ και δεύτερη βάση τη u_2=(11.56, 3.03). Οι καλύτερες λύσεις για κάθε επόμενη κορυφή είναι:

w_1 με $u_1=v_2$ και h_1=3.15 και w_2 με u_2=(11.56, 3.03) και h_2=1.884

w_1 με $u_1=v_3$ και h_1=2.8 και w_2 με $u_2=v_9$ και h_2=1.884

w_1 με $u_1=v_4$ και h_1=6.667 και w_2 με $u_2=v_9$ και h_2=0.0

w_1 με $u_1=v_5$ και $h_1=8.1$ και w_2 με $u_2=v_9$ και $h_2=0.0$

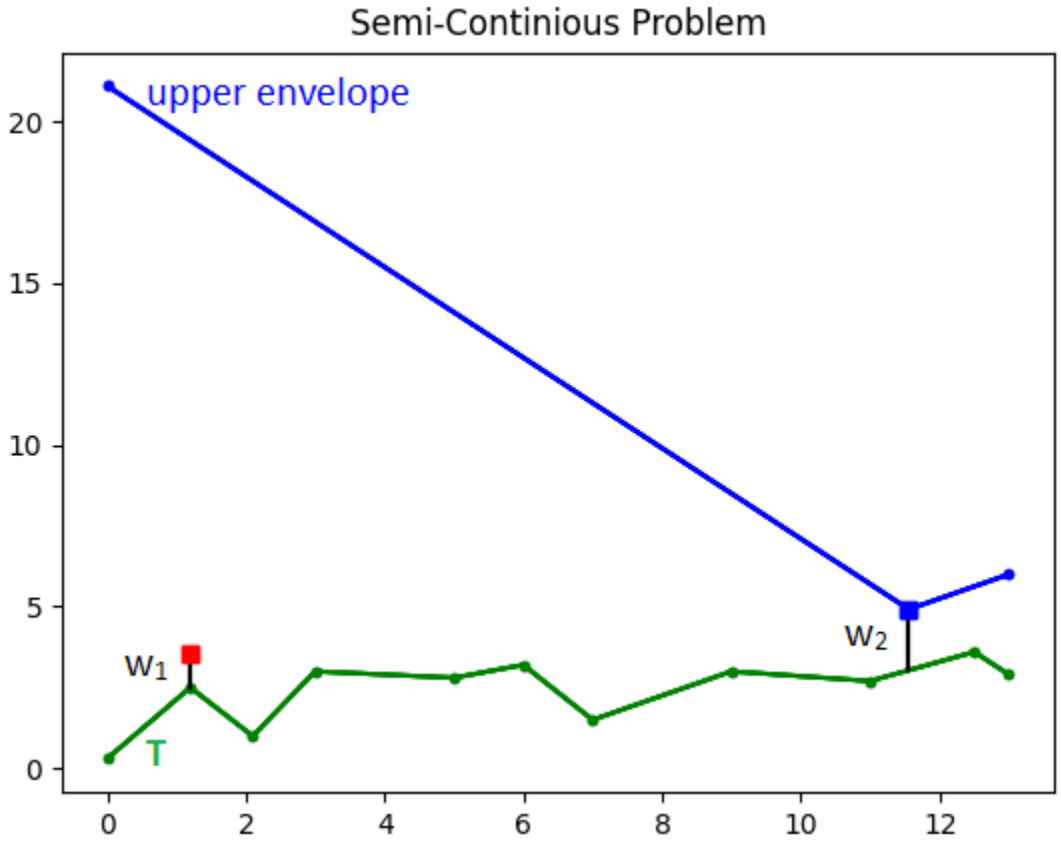
w_1 με $u_1=v_6$ και $h_1=1.8$ και w_2 με $u_2=v_5$ και $h_2=9.5$

w_1 με $u_1=v_7$ και $h_1=5.5$ και w_2 με $u_2=v_1$ και $h_2=0.0$

w_1 με $u_1=v_8$ και $h_1=3$ και w_2 με $u_2=v_1$ και $h_2=0.0$

w_1 με $u_1=v_9$ και $h_1=2.025$ και w_2 με $u_2=v_1$ και $h_2=0.68$

Στο Σχήμα 2.17 φαίνεται το ζεύγος επιτήρησης της T του Σχήματος 2.10 στην ημι-συνεχή εκδοχή του προβλήματος.

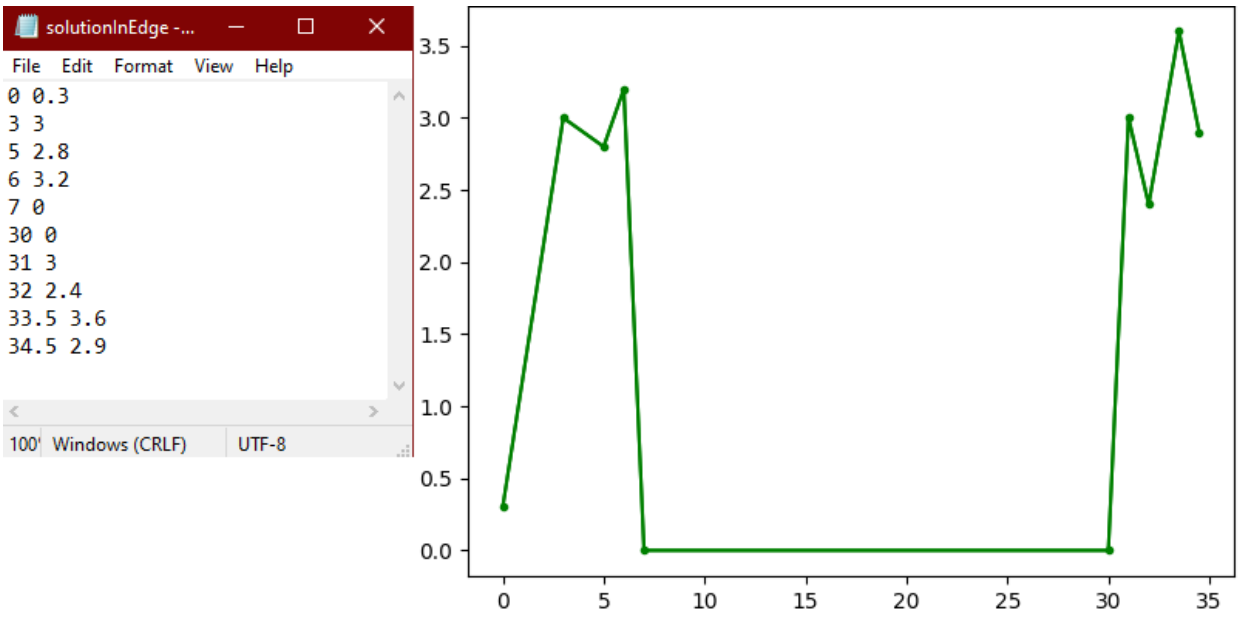


Σχήμα 2.17 *Η T του σχήματος 2.10 και η επιτήρησή της (λύση) στην ημι-συνεχή εκδοχή από τους πύργους w_1 και w_2.*

Κεφάλαιο 3. Η Υλοποίηση

3.1 Είσοδος – Έξοδος

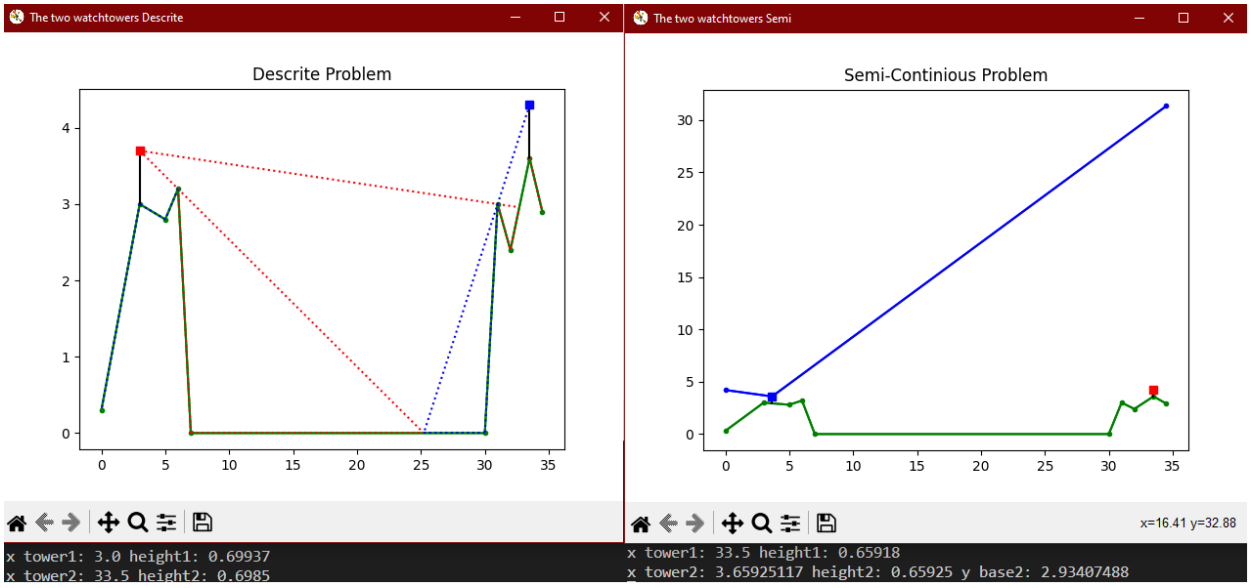
Η είσοδος του προγράμματος που υλοποιήσαμε είναι ένα αρχείο μορφής .txt που περιέχει τις συντεταγμένες των κορυφών της Τ. Συγκεκριμένα, κάθε γραμμή του αρχείου αντιστοιχεί σε μία κορυφή και περιέχει ένα πραγματικό αριθμό που είναι η τετμημένη της κορυφής, κενό και έναν πραγματικό αριθμό που είναι η τεταγμένη της κορυφής (Σχήμα 3.1). Από πάνω προς τα κάτω οι τετμημένες των κορυφών είναι σε αύξουσα σειρά. Πολλά τέτοια αρχεία μορφής .txt μπορούν να είναι αποθηκευμένα στο φάκελο terrains που βρίσκεται στα αρχεία του κώδικα. Έτσι μπορούν να είναι προσβάσιμα από το πρόγραμμα πολλές x-μονότονες πολυγωνικές γραμμές Τ και επίσης έχουμε τη δυνατότητα να τροποποιήσουμε τις τιμές των κορυφών σε ένα αρχείο και να το φορτώσουμε στο πρόγραμμα χωρίς να χρειάζεται να επανεκκινήσουμε τον κώδικα.



Σχήμα 3.1 *Η είσοδος του προγράμματος είναι ένα αρχείο μορφής .txt (αριστερά) και πως μεταφράζεται από το πρόγραμμα (δεξιά).*

Η έξοδος του προγράμματος στην διακριτή εκδοχή του προβλήματος είναι οι τετμημένες των κορυφών-βάσεων των δύο πύργων και τα ύψη τους. Αυτές τις τιμές τις βλέπουμε στο τερματικό που εκτελέσαμε το πρόγραμμα. Επίσης εμφανίζεται και ένα σχήμα (Σχήμα 3.2) που

μπορεί να δει κάποιος καλύτερα την λύση του προβλήματος. Στην ημι-συνεχή εκδοχή του προβλήματος, λόγω της ελευθερίας του δεύτερου πύργου το πρόγραμμα επιστρέφει στο τερματικό τη τετμημένη της κορυφής-βάσης του πρώτου πύργου, τις συντεταγμένες της βάσης του δεύτερου πύργου και τα ύψη τους. Επίσης, όπως και στην διακριτή εκδοχή, εμφανίζεται το παράθυρο με το σχήμα που περιέχει του δύο πύργους με ελάχιστο κοινό ύψος που καλύπτουν την ορατότητα και το upper envelope για να γίνει πιο κατανοητή η επιλογή του δεύτερου πύργου.



Σχήμα 3.2 *Αριστερά οι έξοδοι της διακριτής εκδοχής του προβλήματος και δεξιά οι έξοδοι της ημι-συνεχής εκδοχής του προβλήματος.*

3.2 Δομές Δεδομένων

Ο κώδικας του προγράμματος υλοποιήθηκε στη γλώσσα προγραμματισμού Python στην οποία δομές δεδομένων όπως οι στοίβες και οι πίνακες, που ο αλγόριθμος χρησιμοποιεί κυρίως, αναπαρίστανται ως λίστες. Έτσι όλες οι δομές δεδομένων του προγράμματος είναι λίστες.

3.3 Αρχεία/Διαδικασίες/Συναρτήσεις

Ο κώδικας του προγράμματος είναι χωρισμένος σε τρία αρχεία python: visualization.py, algorithm.py και divideAndConquer.py. Το αρχείο algorithm.py περιέχει κάποιες βοηθητικές

συναρτήσεις που χρησιμοποιούνται πολλαπλές φορές από τον κώδικα. Στο αρχείο divideAndConquer.py υλοποιείται η διαδικασία εύρεσης των ζευγαριών ορατότητας με την μέθοδο διαίρει και βασίλευε. Το αρχείο visualization.py έχει ένα πιο γενικό ρόλο, καθώς υλοποιεί λειτουργίες main, οπτικοποίησης και υλοποίησης των αλγορίθμων της διακριτής και ημι-συνεχούς εκδοχής.

3.3.1 Αρχείο visualization.py

Στο αρχείο visualization.py κατασκευάζουμε ένα γραφικό περιβάλλον όπου η χρήση του αλγορίθμου επιτυγχάνεται διαδραστικά. Πιο συγκεκριμένα κατά την εκκίνηση του προγράμματος εμφανίζεται ένα παράθυρο με διάφορες επιλογές ως κουμπιά με πράσινο ή λευκό χρώμα. Τα πράσινα κουμπιά είναι οι κύριες λειτουργίες του προγράμματος δηλαδή η φόρτωση x-μονότονης πολυγωνικής γραμμής, η εκτέλεση της διακριτής εκδοχής και η εκτέλεση της ημι-συνεχούς εκδοχής. Τα λευκά κουμπιά, είναι η οπτικοποίηση των ενδιάμεσων λειτουργιών του αλγορίθμου για έναν επιλεγμένο πύργο. Σε αυτό το αρχείο χρησιμοποιούμαι κάποιες global μεταβλητές που περιγράφουν την T (xOfVertices,yOfVertices), τον πρώτο πύργο (xTower-τετμημένη βάσης, yBase-τεταγμένη βάσης, yTower-τεταγμένη άνω άκρου, h-ύψος, numberTower-δείκτης κορυφής/βάσης) και την ορατότητα του πρώτου πύργου (setP-το σύνολο P, setVisP-οι οριακά ορατές κορυφές, terrainWithP-η T με τα σημεία P, setNotVisibleEdges-οι μη ορατές ακμές από τον πρώτο πύργο).

3.3.1.1 Διαδικασία initiate()

Η διαδικασία initiate() είναι η πρώτη διαδικασία που εκτελεί το πρόγραμμα και ο ρόλος της είναι η κατασκευή και η εμφάνιση του παραθύρου περιήγησης στις λειτουργίες του αλγορίθμου.

```
def initiate():
    global myTerrains, winLoadTerrain, terrainPath, winVertices, listOfVertices,
text_boxHeight
    terrainPath = os.path.dirname(os.path.abspath(__file__))
    terrains = os.listdir(terrainPath + "\\terrains")
    window = Tk()
    window.title("Two Watchtowers")
```

```
    window.geometry("500x400+1000+300")
    # Top level for loading a terrain

    winLoadTerrain = Toplevel()
    winLoadTerrain.title("Load Terrain")
    winLoadTerrain.geometry("200x210+800+300")
    winLoadTerrain.withdraw()
    winLoadTerrain.protocol("WM_DELETE_WINDOW", disable_event)

    buttonLoadTerrain = Button(winLoadTerrain, text= "Load Terrain", command=
loadTerrain)
    buttonLoadTerrain.place(x=10,y=185)

    sbAvailableVertices = Scrollbar(winLoadTerrain)
    sbAvailableVertices.pack(side = RIGHT, fill = Y)
    myTerrains = Listbox(winLoadTerrain, yscrollcommand = sbAvailableVertices.set)
    myTerrains.pack(side = LEFT)
    sbAvailableVertices.config(command = myTerrains.yview)
    for script in terrains:
        myTerrains.insert(END, script[:-4])

    # Top level for choosing vertex for the first tower

    winVertices = Toplevel()
    winVertices.title("Choose Vertex")
    winVertices.geometry("150x210+850+300")
    winVertices.withdraw()
    winVertices.protocol("WM_DELETE_WINDOW", disable_event)
    buttonOk = Button(winVertices, text= "Ok", command= updateTower)
    buttonOk.place(x=60,y=185)

    sbAvailableVertices = Scrollbar(winVertices)
    sbAvailableVertices.pack(side = RIGHT, fill = Y)
    listOfVertices = Listbox(winVertices, yscrollcommand = sbAvailableVertices.set)
    listOfVertices.pack(side = LEFT)
    sbAvailableVertices.config(command = listOfVertices.yview)
```

```
    text_boxHeight = Text(
    winVertices,
    height=1,
    width=15
    )
    text_boxHeight.place(x=0,y=0)
    text_boxHeight.insert('end', message1)
    text_boxHeight.config(state='normal')


    #Buttons on main window
    buttonChooseTerrain = Button(window, bg='green', text= "Choose Terrain", command=
chooseTerrain)
    buttonChooseVertex = Button(window, text= "Choose Vertex", command= chooseVertex)
    buttonCurrentTerrain = Button(window, text= "Draw Vis Lines", command=
drawVisibilityLines)
    buttonLeftPUV = Button(window, text= "leftPUV", command= calculateLeftPUV)
    buttonRightPUV = Button(window, text= "rightPUV", command= calculateRightPUV)


    buttonFindSecondTower = Button(window, bg='green', text= "Visualize Discrete",
command= visualizeDiscrete)


    buttonCalculateManualSecond = Button(window, text= "Calculate manual Tower",
command= calculateSecondTower)


    buttonCalculateUpperEnvelope = Button(window, text= "Visualize Upper Envelope With
All Pairs", command= visualizeUpperEnvelope)


    buttonCalculateSemiContinious = Button(window, text= "Semi-Continious", command=
calculateSecondTowerSemiContinious)
    buttonCalculateAutoSemiContinious = Button(window, text= "Auto-Semi-Continious",
command= calculateSecondTowerSemiContiniousSolo)


    buttonsVisualizeSemiCOntinious = Button(window, bg='green', text= "Visualize Semi-
Continious", command= visualizeSemicontinious)


    buttonChooseTerrain.place(x = 10, y = 10)
    buttonChooseVertex.place(x = 10 , y = 40)
    buttonCurrentTerrain.place(x = 10, y = 70)
    buttonLeftPUV.place(x = 10,y = 100)
```

```
    buttonRightPUV.place(x = 10, y = 130)
    buttonFindSecondTower.place(x = 10, y = 160)
    buttonCalculateManualSecond.place(x=10,  y=190)
    buttonCalculateUpperEnvelope.place(x=10, y=220)
    buttonCalculateSemiContinious.place(x=10, y=250)
    buttonCalculateAutoSemiContinious.place(x=10, y=280)
    buttonsVisualizeSemiCOntinious.place(x=10,y=310)
    print("hello etiko")
    window.mainloop()
```

3.3.1.2 Διαδικασία loadTerrain()

Η διαδικασία εμφανίζει σε ένα γραφικό περιβάλλον την Τ που επίλεξε ο χρήστης από το παράθυρο περιήγησης και ενημερώνει τις global μεταβλητές της Τ.

```
def loadTerrain():
    global xOfVertices, yOfVertices, xTower, yBase, yTower, numberTower,setP
    try:
        script = myTerrains.get(myTerrains.curselection())
        p = terrainPath+"\\terrains\\" + script + ".txt"
        defaultTerrain = open(p, "r")

        vertices = defaultTerrain.read().splitlines()
        defaultTerrain.close()

        setP = []
        xOfVertices = []
        yOfVertices = []
        xTower = 0
        yBase = 0
        yTower = 0
        numberTower = -1

        for vertice in vertices:
```

```
            coords = vertice.split(" ")
            xOfVertices.append(float(coords[0]))
            yOfVertices.append(float(coords[1]))

        winLoadTerrain.withdraw()
        showTerrain()
    except:
        print("Terrain file moved")
```

3.3.1.3 Διαδικασία drawTerrain()

Η διαδικασία drawTerrain() προετοιμάζει την παρουσίαση μίας Τ και του ενός πρώτου πύργου χωρίς να εμφανίζει κάποιο παράθυρο με γραφικές αναπαραστάσεις.

```
def drawTerrain():
        global xOfVertices,yOfVertices

        plt.figure(num='The two watchtowers')
        plt.title("General terrain")
        plt.plot(xOfVertices,yOfVertices,color = 'green')
        plt.plot(xOfVertices, yOfVertices, marker = ".", color = "green")
        if numberTower != -1:
            plt.plot([xTower,xTower],[yBase,yTower],color = '0')
            plt.plot([xTower],[yTower], marker = "s")
        plt.draw()
```

3.3.1.4 Διαδικασία showTerrain()

Η διαδικασία showTerrain() κλείνει όλες τις γραφικές αναπαραστάσεις και εμφανίζει ότι κατασκεύασε η drawTerrain().

```
def showTerrain():
       plt.close()
       plt.close()
```

```
        plt.close()
        drawTerrain()
        plt.show()
```

3.3.1.5 Διαδικασία updateTower()

Η διαδικασία updateTower() εμφανίζει στην επιλεγμένη T ένα πύργο για τον οποίο επιλέγουμε στο παράθυρο περιήγησης το ύψος και την κορυφή από την οποία θα υψωθεί. Επίσης, ενημερώνει τις global μεταβλητές του πρώτου πύργου.

```
def updateTower():
    global xTower, yBase, yTower, numberTower,h,setP
    setP = []

    try:
        xTower = float(listOfVertices.get(listOfVertices.curselection()))
        numberTower = xOfVertices.index(xTower)
        yBase = yOfVertices[numberTower]
        h = float(text_boxHeight.get("1.0",END))
        yTower = yBase + h
        winVertices.withdraw()

    except:
        print("Not valid vertice or height")
    showTerrain()
```

3.3.1.5 Διαδικασία autoUpdateTower(x,y)

Η διαδικασία autoUpdateTower(x,y) δέχεται ως είσοδο την τετμημένη μία κορυφής και μία τεταγμένη και ενημερώνει τις global μεταβλητές του πρώτου πύργου.

```
def autoUpdateTower(x,y):
    global xTower, yBase, yTower, numberTower,h,setP
    setP = []
```

```
    xTower = x
    yTower = y
    numberTower = xOfVertices.index(xTower)
    yBase = yOfVertices[numberTower]
    h = yTower - yBase
```

3.3.1.6 Διαδικασία chooseTerrain()

Η διαδικασία chooseTerrain() εμφανίζει το παράθυρο από το οποίο επιλέγει ο χρήστης την Τ.

```
def chooseTerrain():
    winLoadTerrain.deiconify()
```

3.3.1.7 Διαδικασία chooseVertex()

Η διαδικασία chooseVertex() εμφανίζει και ανανεώνει ανάλογα την Τ που επιλέχθηκε τις κορυφές από τις οποίες ο χρήστης θα επιλέξει να υψώσει ένα πρώτο πύργο.

```
def chooseVertex():
    global winVertices, listOfVertices, text_boxHeight
    listOfVertices.delete(0,END)
    winVertices.deiconify()
    for x in xOfVertices:
        listOfVertices.insert(END,  str(x))
```

3.3.1.8 Διαδικασία autoCalculateVisibilityLines()

Η διαδικασία autoCalculateVisibilityLines() ενημερώνει τις global μεταβλητές ορατότητας ανάλογα τον πρώτο πύργο.

```
def autoCalculateVisibilityLines():
    # active input: nothing
```

```
    # passive input: terrain T as global
    # outputs(all global):
    #          setP - p points+ not visible vertices
    #          terrainWithP - T + p points
    #          setNotVisibleEdges - all invisible edges and partly invisiblle edges
    #          setVisP - p points

    global setP, terrainWithP ,setNotVisibleEdges,setVisP
    setNotVisibleEdges = []

    endPoints = alg.visibilityEndpoints(xOfVertices, yOfVertices, xTower, yTower)
    #returns [[p points- [x,y]], [endpoints and vertices out of sight-
[x,y]],[vertices in sight where visibility ends-[x,y]]

    setPonVertice= []
    for endPoint in endPoints[0]:
        if endPoint[0] in xOfVertices:
            setPonVertice.append(endPoint)
    setP = endPoints[0]
    setVisP = copy.deepcopy(setP)
    for i in range(len(setP)):
        setP[i][0] = round(setP[i][0],13)
        setP[i][1] = round(setP[i][1],13)

    #calculate terrainWithP  -avoid double vertices
    terrainWithP = [xOfVertices.copy(),yOfVertices.copy()]
    setPwithNoVertice = copy.deepcopy(setP)
    epCounter = 0
    if len(setPwithNoVertice)>0:
        for i in range(len(xOfVertices)):
            if len(setPwithNoVertice)>0:
                if epCounter >= len(setPwithNoVertice):
                        break
                if  xOfVertices[i] == setPwithNoVertice[epCounter][0]:
                    setPwithNoVertice.remove(setPwithNoVertice[epCounter])
                    continue
```

```
                if  xOfVertices[i] > setPwithNoVertice[epCounter][0]:

terrainWithP[0].insert(epCounter+i,setPwithNoVertice[epCounter][0])

terrainWithP[1].insert(epCounter+i,setPwithNoVertice[epCounter][1])
                    epCounter+=1


    for p in setPonVertice:
        endPoints[1].remove(p)


    #calculate not visible edges in setNotVisibleEdges
    for i in range (len(endPoints[1])-1):
        if endPoints[1][i][0] in xOfVertices and endPoints[1][i+1][0] in xOfVertices:
            start = xOfVertices.index(endPoints[1][i][0])
            finish = xOfVertices.index(endPoints[1][i+1][0])
            if abs(start - finish) == 1:
                setNotVisibleEdges.append([endPoints[1][i],endPoints[1][i+1]])
        if endPoints[1][i][0] in xOfVertices and endPoints[1][i+1] in setP and
endPoints[1][i+1] not in setPonVertice:
            setNotVisibleEdges.append([endPoints[1][i],endPoints[1][i+1]])



    #create setP (it works so I don't touch it)
    #In the beggining I wanted only p points then I added all fully invisible V + p-
points
    setPnotValuable = copy.deepcopy(setPonVertice)
    for point in setPnotValuable:
        counter = 0
        for pair in setNotVisibleEdges:
            for p in pair:
                if p == point:
                    counter+=1
        if counter==1:
            setPnotValuable.remove(point)
    for p in setP:
        if p in setPnotValuable:
            setP.remove(p)
```

```
    for hiddenPiece in setNotVisibleEdges:
        for v in hiddenPiece:
            if v not in setP and v not in endPoints[2]:
                setP.append(v)
    setP = sorted(setP, key=lambda p:p[0])
```

3.3.1.9 Συνάρτηση autoCalculateLeftPUV()

Η συνάρτηση autoCalculateLeftPUV υπολογίζει και επιστρέφει τα αριστερά ζεύγη ορατότητας με δεδομένα τις global μεταβλητές.

```
def autoCalculateLeftPUV():
    puv = dc.constructPUV(xOfVertices, terrainWithP,setP)
    setE = []
    for part in setNotVisibleEdges:
        setE.append(part)
    return [puv,setE]
```

3.3.1.10 Συνάρτηση autoCalculateRightPUV()

Η συνάρτηση autoCalculateRightPUV υπολογίζει και επιστρέφει τα δεξιά ζεύγη ορατότητας με δεδομένα τις global μεταβλητές.

```
def autoCalculateRightPUV():
    # idea is to mirror terrainWithP and setP and then calulate left PV pairs in the
mirrored terrainWithP
    # afte mirroring the outputs you get right PV pairs
    rightTerrain = copy.deepcopy(terrainWithP)

    rightSetP = copy.deepcopy(setP)
    rightTerrain[0] = [rightTerrain[0][-1] - x for x in rightTerrain[0]]

    for i in range(len(rightSetP)):
        rightSetP[i][0] = terrainWithP[0][-1] - rightSetP[i][0]
```

```
    rightTerrain[0].reverse()
    rightTerrain[1].reverse()
    rightSetP.reverse()
    revXofVertices = copy.deepcopy(xOfVertices)
    revXofVertices = [xOfVertices[-1] - x for x in xOfVertices]
    rightPUV = dc.constructPUV(revXofVertices, rightTerrain,rightSetP)
    for i in range(len(rightPUV)):
        rightPUV[i][0][0] = xOfVertices[-1]-rightPUV[i][0][0]
        rightPUV[i][1][0] = xOfVertices[-1]-rightPUV[i][1][0]
    setE = []
    for part in setNotVisibleEdges:
        setE.append(part)
    rightPV = [rightPUV,setE]
    return rightPV
```

3.3.1.11 Συνάρτηση calculateSecondTowerDiscreteSolo()

Η συνάρτηση calculateSecondTowerDiscreteSolo() με την χρήση των global μεταβλητών της T υπολογίζει και επιστρέφει τη λύση του διακριτού προβλήματος.

```
#inputs:globals xOfVertices,yOfVertices
#outputs: shortest common height pair of towers
def calculateSecondTowerDiscreteSolo():
    edges = []
    for i in range(len(xOfVertices)-1):

edges.append([[xOfVertices[i],yOfVertices[i]],[xOfVertices[i+1],yOfVertices[i+1]]])
    tower1 = -1
    tower2 = -1
    height1 = sys.float_info.max
    height2 = sys.float_info.max
    for i in range(len(xOfVertices)-1):
        setE = edges
        setA = []
```

```
        criticalPairs = calculateCriticalHeights(xOfVertices[i])
        for pair in criticalPairs:
            setA.append(pair)
        interEdgeTower = []
        towerLine =
[[xOfVertices[i],yOfVertices[i]],[xOfVertices[i],yOfVertices[i]+2]]
        for line in setE:
            temp = alg.intersectionPoint(line ,towerLine)[1]
            if temp >= yOfVertices[i]:
                interEdgeTower.append(temp)
        for line in setA:
            temp = alg.intersectionPoint(line ,towerLine)[1]
            if temp >= yOfVertices[i]:
                interEdgeTower.append(temp)
        interEdgeTower.sort()
        setL = copy.deepcopy(interEdgeTower)
        left = 0
        right = len(setL)-1
        currentTower2 = -1
        currentHeight2 = sys.float_info.max
        found = FALSE
        iBestCoverTower = -1
        iBestCoverHeight = sys.float_info.max
        iBestHeight = sys.float_info.max
        while not found and left <= right:
            middle = (left + right)//2
            autoUpdateTower(xOfVertices[i],setL[middle])
            currentTower1 = xOfVertices[i]
            currentHeight1 = h
            bestTower =  calculateCoverTower()

            currentTower2 = bestTower[0]
            currentHeight2 = bestTower[1]

            if max(currentHeight1,currentHeight2) < max(iBestHeight,iBestCoverHeight):
                iBestCoverTower = currentTower2
```

```
                iBestCoverHeight = currentHeight2
                iBestHeight = currentHeight1


            if currentHeight1 < currentHeight2:
                left = middle+1
            elif currentHeight1 > currentHeight2:
                right = middle -1
            else:
                found = TRUE
        xwinner = i     #xOfVertices.index(tower1)
        x2winner = xOfVertices.index(iBestCoverTower)
        if iBestHeight!=iBestCoverHeight:
            currentBest =
doubleCheckWinnerDiscrete(xwinner,x2winner,iBestHeight,iBestCoverHeight)
            currentTower1 = currentBest[0]
            currentTower2 = currentBest[1]
            currentHeight1 = currentBest[2]
            currentHeight2 = currentBest[3]
        if max(currentHeight1,currentHeight2) < max(height1,height2):
                tower1 = currentTower1
                tower2 = currentTower2
                height1 = round(currentHeight1,5)
                height2 = round(currentHeight2,5)
        print(xOfVertices[i], tower1,tower2,height1,height2)
    print("x tower1:",tower1, "height1:", height1)
    print("x tower2:",tower2,"height2:", height2)
    return [[tower1,height1],[tower2,height2]]
```

3.3.1.12 doubleCheckWinnerDiscrete(i,j,height1,height2)

Η συνάρτηση doubleCheckWinnerDiscrete(i,j,height1,height2) παίρνει ως είσοδο τις θέσεις ενός πιθανού ζεύγους πύργων τα ύψη τους και επιστρέφει τη βελτιστοποιημένη λύση για τη διακριτή εκδοχή.

```
def doubleCheckWinnerDiscrete(i,j,height1,height2):
    tower1 = xOfVertices[i]
    tower2 = xOfVertices[j]
    height1 = height1
    height2 = height2
    if height1>height2:
        dh = height1-height2
        low = height2
        high = height1
    else:
        dh = height2-height1
        low = height1
        high = height2
    step = dh/10000
    setL = np.arange(yOfVertices[i]+low, yOfVertices[i]+high, step).tolist()
    left = 0
    right = len(setL)-1
    currentTower2 = -1
    currentHeight2 = sys.float_info.max
    found = FALSE
    while not found and left <= right:
        middle = (left + right)//2
        autoUpdateTower(xOfVertices[i],setL[middle])
        currentTower1 = xOfVertices[i]
        currentHeight1 = h
        bestTower =  calculateCoverTower()
        currentTower2 = bestTower[0]
        currentHeight2 = bestTower[1]
        if max(currentHeight1,currentHeight2) < max(height1,height2):
            tower1 = currentTower1
            tower2 = currentTower2
            height1 = currentHeight1
            height2 = currentHeight2
        elif max(currentHeight1,currentHeight2) == max(height1,height2):
            if min(currentHeight1,currentHeight2) < min(height1,height2):
                tower1 = currentTower1
```

```
                tower2 = currentTower2
                height1 = currentHeight1
                height2 = currentHeight2
        if currentHeight1 < currentHeight2:
            left = middle+1
        elif currentHeight1 > currentHeight2:
            right = middle -1
        else:
            found = TRUE
    return [tower1,tower2,height1,height2]
```

3.3.1.13 Συνάρτηση calculateCoverTower()

Η συνάρτηση calculateCoverTower() σύμφωνα με τις global μεταβλητές υπολογίζει τον συμπληρωματικό πύργο στη διακριτή εκδοχή και τον επιστρέφει.

```
def calculateCoverTower():
    global setP
    autoCalculateVisibilityLines()
    for p in setP:
        if p[0]< numberTower+1:
            setP.remove(p)
    visibilityPairs = autoCalculateLeftPUV()
    leftPairsPV = visibilityPairs[0]
    setB = []
    setEv = visibilityPairs[1]
    currentTower2 = 0
    currentHeight2 = sys.float_info.max
    for j in range(numberTower+1,len(xOfVertices)):

        secondTowerLine =
[[xOfVertices[j],yOfVertices[j]],[xOfVertices[j],yOfVertices[j]+2]]
        possibleHeights2 = []

        for line in setB+setEv:
```

```
            if line[0][0] != line[1][0]:
                temp = alg.intersectionPoint(line ,secondTowerLine)[1]
                if round(temp,10) >= yOfVertices[j]:
                    possibleHeights2.append(round(temp,10))
        if len(possibleHeights2) > 0:
            currentY2 = max(possibleHeights2)
            tempHeight2 = currentY2 - yOfVertices[j]
        else:
            tempHeight2 = sys.float_info.max
        if tempHeight2 < currentHeight2:
            currentHeight2 = tempHeight2
            currentTower2 = j
        #sorting is difficult
        for pair in leftPairsPV:
            if pair[0][0] == xOfVertices[j] or pair[1][0] == xOfVertices[j]:
                setB.append(pair)
    if currentHeight2 == sys.float_info.max:
        currentHeight2 = 0
    return [xOfVertices[currentTower2],currentHeight2]
```

3.3.1.14 Συνάρτηση calculateSecondTowerSemiContiniousSolo()

Η συνάρτηση calculateSecondTowerSemiContiniousSolo() με χρήση των global μεταβλητών υπολογίζει και επιστρέφει τη λύση του ημι-συνεχούς προβλήματος.

```
#inputs:globals xOfVertices,yOfVertices
#outputs: shortest common height pair of towers
def calculateSecondTowerSemiContiniousSolo(): #n^3(log^2 n)
    edges = []
    for i in range(len(xOfVertices)-1):

edges.append([[xOfVertices[i],yOfVertices[i]],[xOfVertices[i+1],yOfVertices[i+1]]])
    tower1 = -1
    base1 = -1
    tower2 = -1
```

```
    base2 = -1
    height1 = sys.float_info.max
    height2 = sys.float_info.max
    upperEnvelope = []
    for i in range(len(xOfVertices)-1):
        setE = []
        for e in edges:
            setE.append(e)
        setA = []
        criticalPairs = calculateCriticalHeights(xOfVertices[i])
        for pair in criticalPairs:
            setA.append(pair)
        interEdgeTower = []
        towerLine =
[[xOfVertices[i],yOfVertices[i]],[xOfVertices[i],yOfVertices[i]+2]]
        for line in setE:
            temp = alg.intersectionPoint(line ,towerLine)[1]
            if temp >= yOfVertices[i]:
                interEdgeTower.append(temp)
        for line in setA:
            temp = alg.intersectionPoint(line ,towerLine)[1]
            if temp >= yOfVertices[i]:
                interEdgeTower.append(temp)
        interEdgeTower.sort()
        setL = copy.deepcopy(interEdgeTower)
        left = 0
        right = len(setL)-1
        currentTower2 = -1
        currentBase2 = -1
        currentHeight2 = sys.float_info.max
        found = FALSE
        iBestTower = -1
        iBestCoverTower = -1
        iBestCoverHeight = sys.float_info.max
        iBestHeight = sys.float_info.max
        iBestEnvelope = []
```

```
        iBestBase = -1
        iBestCoverBase = -1
        while not found and left <= right:
            middle = (left + right)//2
            autoUpdateTower(xOfVertices[i],setL[middle])
            currentTower1 = xOfVertices[i]
            currentBase1 = yOfVertices[i]
            currentHeight1 = round(h,8)
            bestTower =  calculateSecondTowerSemiContinious()
            currentTower2 = bestTower[0]
            currentBase2 = bestTower[1]
            currentHeight2 = bestTower[2]
            currentUpperEnvelope = bestTower[3]
            if max(currentHeight1,currentHeight2) < max(iBestHeight,iBestCoverHeight):
                iBestTower = currentTower1
                iBestCoverTower = currentTower2
                iBestBase = currentBase1
                iBestCoverBase = currentBase2
                iBestHeight = currentHeight1
                iBestCoverHeight = currentHeight2
                iBestEnvelope = currentUpperEnvelope

            if currentHeight1 < currentHeight2:
                left = middle+1
            elif currentHeight1 > currentHeight2:
                right = middle -1
            else:
                found = TRUE
        xwinner = i     #xOfVertices.index(tower1)
        x2winner = iBestCoverTower
        if iBestHeight!=iBestCoverHeight:
            currentBest =
doubleCheckWinnerSemiContinious(xwinner,x2winner,iBestHeight,iBestCoverHeight,iBestBas
e,iBestCoverBase,iBestEnvelope)
            currentTower1 = currentBest[0][0]
            currentBase1 = currentBest[0][1]
```

```
            currentHeight1 = currentBest[0][2]
            currentTower2 = currentBest[1][0]
            currentBase2 = currentBest[1][1]
            currentHeight2 = currentBest[1][2]
            currentUpperEnvelope = currentBest[2]
        if max(currentHeight1,currentHeight2) < max(height1,height2):
            tower1 = currentTower1
            tower2 = currentTower2
            base1 = currentBase1
            base2 = currentBase2
            height1 = round(currentHeight1,5)
            height2 = round(currentHeight2,5)
            upperEnvelope = currentUpperEnvelope
        print(xOfVertices[i], tower1,tower2,height1,height2)
    print("x tower1:",tower1, "height1:", height1)
    print("x tower2:",tower2,"height2:", height2, "y base2:",base2)
    return([[tower1,base1,height1],[tower2,base2,height2],upperEnvelope])
```

3.3.1.15 Συνάρτηση doubleCheckWinnerSemiContinious(I, j, height1, height2, base1, base2, envelope)

Η συνάρτηση doubleCheckWinnerSemiContinious(i, j, height1, height2, base1, base2, envelope) παίρνει ως όρισμα τις συντεταγμένες των βάσεων τα ύψη και το upper envelope από το οποίο προκύπτει η λύση της ημι-συνεχούς εκδοχής και επιστρέφει τη βελτιστοποιημένη λύση με τα παραπάνω στοιχεία.

```
def doubleCheckWinnerSemiContinious(i,j, height1, height2, base1, base2, envelope):
    tower1 = xOfVertices[i]
    tower2 = j
    height1 = height1
    height2 = height2
    base1 = base1
    base2 = base2
    upperEnvelope = envelope
    if height1>height2:
```

```
    dh = height1-height2
    low = height2
    high = height1
else:
    dh = height2-height1
    low = height1
    high = height2
step = dh/10000
setL = np.arange(yOfVertices[i]+low, yOfVertices[i]+high, step).tolist()
left = 0
right = len(setL)-1
currentTower2 = -1
currentHeight2 = sys.float_info.max
found = FALSE
while not found and left <= right:
    middle = (left + right)//2
    autoUpdateTower(xOfVertices[i],setL[middle])
    currentTower1 = xOfVertices[i]
    currentBase1 = yOfVertices[i]
    currentHeight1 = round(h,8)
    bestTower =  calculateSecondTowerSemiContinious()
    currentTower2 = bestTower[0]
    currentBase2 = bestTower[1]
    currentHeight2 = bestTower[2]
    currentUpperEnvelope = bestTower[3]
    if max(currentHeight1,currentHeight2) < max(height1,height2):
        tower1 = currentTower1
        tower2 = currentTower2
        base1 = currentBase1
        base2 = currentBase2
        height1 = currentHeight1
        height2 = currentHeight2
        upperEnvelope = currentUpperEnvelope
    elif max(currentHeight1,currentHeight2) == max(height1,height2):
        if min(currentHeight1,currentHeight2) < min(height1,height2):
            tower1 = currentTower1
```

```
                tower2 = currentTower2
                base1 = currentBase1
                base2 = currentBase2
                height1 = currentHeight1
                height2 = currentHeight2
                upperEnvelope = currentUpperEnvelope
        if currentHeight1 < currentHeight2:
            left = middle+1
        elif currentHeight1 > currentHeight2:
            right = middle -1
        else:
            found = TRUE
    return([[tower1,base1,height1],[tower2,base2,height2],upperEnvelope])
```

3.3.1.16 Συνάρτηση calculateSecondTowerSemiContinious()

Η συνάρτηση calculateSecondTowerSemiContinious() υπολογίζει σύμφωνα με τις global μεταβλητές και επιστρέφει τον δεύτερο κοντύτερο πύργο που μαζί με τον πρώτο καλύπτουν την ορατότητα στην ημι-συνεχή εκδοχή καθώς και το upper envelope από το οποίο προκύπτει η λύση.

```
#input: global first tower
#output: shortest cover Tower
def calculateSecondTowerSemiContinious():
    autoCalculateVisibilityLines() #O(n)
    visibilityPairsLeft = autoCalculateLeftPUV() #O(n^2logn)
    visibilityPairsRight = autoCalculateRightPUV() #O(n^2logn)
    setB = []
    setC = visibilityPairsRight[0]
    for pair in setC:
        if pair[0][0] == xOfVertices[0] or pair[1][0] == xOfVertices[0]:
            setC.remove(pair)
    setNotVisE = visibilityPairsRight[1]
    currentTower2 = 0
    currentBase2 = 0
```

```
    currentHeight2 = sys.float_info.max
    currentUpE =[]
    edges = []
    setLines = setB+setC+setNotVisE
    upE = calculateUpperEnvelope(setLines)
    possibleHeights2 = []
    secondTowerLine = [[xOfVertices[0],yOfVertices[0]],[xOfVertices[0],yOfVertices[0]+2]]
    for line in setLines:
        if line[0][0] != line[1][0]:
            temp = alg.intersectionPoint(line ,secondTowerLine)[1]
            if round(temp,10) >= yOfVertices[0]:
                possibleHeights2.append(round(temp,8))
    if len(possibleHeights2) > 0:
        currentY2 = max(possibleHeights2)
        tempHeight2 = currentY2 - yOfVertices[0]
        tempBase2 = yOfVertices[0]
    else:
        tempHeight2 = sys.float_info.max
    if tempHeight2 < currentHeight2:
        currentHeight2 = round(tempHeight2,8)
        currentTower2 = xOfVertices[0]
        currentBase2 = tempBase2
        currentUpE = upE
    for i in range(len(xOfVertices)-1):
edges.append([[xOfVertices[i],yOfVertices[i]],[xOfVertices[i+1],yOfVertices[i+1]]])
    for e in edges:
        setLines = setB+setC+setNotVisE
        upE = calculateUpperEnvelope(setLines) #n^2logn
        if upE == 0: #if first tower sees all
            return [0,0,0,[]]
        for k in range(len(upE[0])):
            if upE[0][k] >= e[0][0] and upE[0][k] <= e[1][0]:
                secondTowerLineInUpEnv = [[upE[0][k],upE[1][k]],[upE[0][k],upE[1][k]+1]]
                tempBase = alg.intersectionPoint(e ,secondTowerLineInUpEnv)[1]
```

```
            tempHeight2 = round(upE[1][k] - tempBase,8)
            if tempHeight2<0:
                tempHeight2=0
            if tempHeight2 < currentHeight2:
                currentHeight2 = round(tempHeight2,8)
                currentTower2 = round(upE[0][k],8)
                currentBase2 = round(tempBase,8)
                currentUpE = upE

    for pair in visibilityPairsLeft[0]:
        if pair[0][0] == e[1][0] or pair[1][0] == e[1][0]:
            setB.append(pair)
    setLines = setB+setC+setNotVisE
    possibleHeights2 = []
    secondTowerLine = [[e[1][0],e[1][1]],[e[1][0],e[1][1]+2]]
    for line in setLines:
        if line[0][0] != line[1][0]:
            temp = alg.intersectionPoint(line ,secondTowerLine)[1]
            if round(temp,8) >= e[1][1]:
                possibleHeights2.append(round(temp,10))
    if len(possibleHeights2) > 0:
        currentY2 = max(possibleHeights2)
        tempHeight2 = currentY2 - e[1][1]
        if tempHeight2<0:
                tempHeight2=0
        tempBase2 = e[1][1]
        if tempHeight2 < currentHeight2:
            currentHeight2 = tempHeight2
            currentTower2 = e[1][0]
            currentBase2 = e[1][1]
            currentUpE = upE
return [currentTower2,currentBase2,currentHeight2,currentUpE]
```

3.3.1.17 Συνάρτηση calculateUpperEnvelope(setLines)

Η σύνάρτηση calculateUpperEnvelope(setLines) που έχει ως είσοδο τα ευθύγραμμα τμήματα των οποίων οι φορείς σχηματίζουν το επιθυμητό upper envelope και το επιστρέφει. Επίσης, μορφοποιεί το upper envelope έτσι ώστε να βρίσκεται στα όρια της T και να μην κλιμακώνει τις γραφικές παραστάσεις που κατασκευάζουμε με ανεπιθύμητο τρόπο.

```
def calculateUpperEnvelope(setLines):

    upperEnvelope = [[],[]]

    upperEnvelopeLines = []

    #sort lines based on slope

    setLines = sorted(setLines, key=lambda line: (line[1][1] - line[0][1]) /
(line[1][0] - line[0][0]))

    slopes = []

    for i in range(len(setLines)):

        slopes.append(round((setLines[i][0][1]-setLines[i][1][1])/(setLines[i][0][0]-
setLines[i][1][0]),8))

    slopes.sort()

    #remove parallel Lines

    i = 0

    while i < (len(slopes)-1):


        if slopes[i] == slopes[i+1]:

            if slopes[i]>0:

                if setLines[i][0][0] < setLines[i+1][0][0]:

                    setLines.pop(i+1)

                    slopes.pop(i+1)

                else:

                    setLines.pop(i)

                    slopes.pop(i)

            elif slopes[i]==0:

                if setLines[i][0][1] < setLines[i+1][0][1]:

                    setLines.pop(i)

                    slopes.pop(i)

                else:

                    setLines.pop(i+1)

                    slopes.pop(i+1)

            else:
```

```
                if setLines[i][0][0] > setLines[i+1][0][0]:
                    setLines.pop(i+1)
                    slopes.pop(i+1)
                else:
                   setLines.pop(i)
                   slopes.pop(i)
        else:
            i+=1
    #catch special cases
    if len(setLines)==0: #all visible
        return 0
    elif len(setLines)==1: #not visible edges with same slope
        temp =
alg.intersectionPoint([[xOfVertices[0],yOfVertices[0]],[xOfVertices[0],yOfVertices[0]+
1]],setLines[0])
        upperEnvelope[0].insert(0,temp[0])
        upperEnvelope[1].insert(0,temp[1])
        temp = alg.intersectionPoint([[xOfVertices[-1],yOfVertices[-1]],[xOfVertices[-
1],yOfVertices[-1]+1]],setLines[0])
        upperEnvelope[0].append(temp[0])
        upperEnvelope[1].append(temp[1])
        return upperEnvelope
    #calculate Upper Envelope raw
    temp = alg.intersectionPoint(setLines[0],setLines[1])
    upperEnvelope[0].append(temp[0])
    upperEnvelope[1].append(temp[1])
    upperEnvelopeLines.append(setLines[0])
    upperEnvelopeLines.append(setLines[1])
    for line in setLines[2:]:
        flag = TRUE
        while len(upperEnvelopeLines) > 0 and flag :
            temp = alg.intersectionPoint(line,upperEnvelopeLines[-1])
            if len(upperEnvelope[0]) == 0:
                break
            if temp[0] <= upperEnvelope[0][-1] :
                upperEnvelopeLines.pop()
                upperEnvelope[0].pop()
```

```
                upperEnvelope[1].pop()
            else:
                flag = FALSE

        upperEnvelope[0].append(temp[0])
        upperEnvelope[1].append(temp[1])
        upperEnvelopeLines.append(line)
    #shape upper envelope to the xdimensions of T
    if len(upperEnvelope[0])>1:
        while upperEnvelope[0][1] < xOfVertices[0]:
            upperEnvelope[0].pop(0)
            upperEnvelope[1].pop(0)
    if upperEnvelope[0][0]>xOfVertices[0]:
        temp =
alg.intersectionPoint([[xOfVertices[0],yOfVertices[0]],[xOfVertices[0],yOfVertices[0]+
1]],upperEnvelopeLines[0])
        upperEnvelope[0].insert(0,temp[0])
        upperEnvelope[1].insert(0,temp[1])
    else:
        if len(upperEnvelope[0])>1:
            print(upperEnvelope)
            tempLine =
[[upperEnvelope[0].pop(0),upperEnvelope[1].pop(0)],[upperEnvelope[0][0],upperEnvelope[
1][0]]]
            temp =
alg.intersectionPoint([[xOfVertices[0],yOfVertices[0]],[xOfVertices[0],yOfVertices[0]+
1]],tempLine)
            upperEnvelope[0].insert(0,temp[0])
            upperEnvelope[1].insert(0,temp[1])
    if len(upperEnvelope[0])>1:
        while upperEnvelope[0][-2] > xOfVertices[-1]:
            upperEnvelope[0].pop()
            upperEnvelope[1].pop()
    if upperEnvelope[0][-1]<xOfVertices[-1]:
        temp = alg.intersectionPoint([[xOfVertices[-1],yOfVertices[-1]],[xOfVertices[-
1],yOfVertices[-1]+1]],upperEnvelopeLines[-1])
        upperEnvelope[0].append(temp[0])
        upperEnvelope[1].append(temp[1])
    else:
```

```
        tempLine =
[[upperEnvelope[0].pop(),upperEnvelope[1].pop()],[upperEnvelope[0][-
1],upperEnvelope[1][-1]]]

        temp = alg.intersectionPoint([[xOfVertices[-1],yOfVertices[-1]],[xOfVertices[-
1],yOfVertices[-1]+1]],tempLine)

        upperEnvelope[0].append(temp[0])

        upperEnvelope[1].append(temp[1])

    #shape in case first or last line is under T meaning whole edges of T are visible

    if upperEnvelope[1][-1]< yOfVertices[-1]:

        upperEnvelope[0].pop()

        upperEnvelope[1].pop()

        for i in range(len(xOfVertices)):

            if xOfVertices[i]> upperEnvelope[0][-1]:

                upperEnvelope[0].append(xOfVertices[i])

                upperEnvelope[1].append(yOfVertices[i])

    if upperEnvelope[1][0]< yOfVertices[0]:

        upperEnvelope[0].pop(0)

        upperEnvelope[1].pop(0)

        for i in range(len(xOfVertices)):

            if xOfVertices[i] < upperEnvelope[0][-1]:

                upperEnvelope[0].append(xOfVertices[i])

                upperEnvelope[1].append(yOfVertices[i])


    return(upperEnvelope)
```

3.3.1.18 Συνάρτηση calculateCriticalHeights(tower)

Η συνάρτηση calculateCriticalHeights(tower) με είσοδο την τετμημένη της βάσης του πρώτου πύργου υπολογίζει και επιστρέφει τα κρίσιμα ύψη.

```
def calculateCriticalHeights(tower):
    xvl = []
    yvl = []
    xvr = []
    yvr = []
    rch = []
```

```
    lch = []
    for i in range(len(xOfVertices)):
        if xOfVertices[i] <= tower:
            xvl.append(xOfVertices[i])
            yvl.append(yOfVertices[i])
        if xOfVertices[i] >= tower:
            xvr.append(xOfVertices[i])
            yvr.append(yOfVertices[i])
    xvl.reverse()
    yvl.reverse()
    if len(xvr)>3:
        rch = calculateCriticalPairs(copy.deepcopy(xvr),copy.deepcopy(yvr),1)
    if len(xvl)>3:
        lch = calculateCriticalPairs(copy.deepcopy(xvl),copy.deepcopy(yvl),-1)
    i=0
    while i < len(rch):
        flag = 0
        for j in range(len(xvr)):

            if xvr[j]> rch[i][1][0]:
                break
            if alg.intersectionPoint(rch[i],[[xvr[j],yvr[j]],[xvr[j],yvr[j]+1]])[1] <
yvr[j]-0.00000001:
                flag =1
                z=rch.pop(i)
                break
        if flag == 0:
            i+=1
    i=0
    while i < len(lch):
        flag = 0
        for j in range(len(xvl)):

            if xvl[j]< lch[i][1][0]:
                break
```

```
            if alg.intersectionPoint(lch[i],[[xvl[j],yvl[j]],[xvl[j],yvl[j]+1]])[1] <
yvl[j]-0.00000001:
                flag =1
                z=lch.pop(i)
                break
        if flag == 0:
            i+=1
    return rch+lch
```

3.3.1.19 Συνάρτηση calculateCriticalPairs(xv,yv,t)

Η συνάρτηση calculateCriticalPairs(xv,yv,t) δέχεται ως όρισμα την τετμημένη και την τεταγμένη της βάσης του πρώτου πύργου και αν ψάχνει φορείς που δημιουργούν κρίσιμα ύψη αριστερά ή δεξιά αυτής και επιστρέφει τους φορείς.

```
def calculateCriticalPairs(xv,yv,t):
    yVerticeStart = yv[0]
    xVerticeStart = xv[0]
    towerLine = [[xv[0],yVerticeStart],[xv[0],yVerticeStart+2]]
    criticalPairs = []
    convexHull = []
    convexHull.append([xv.pop(0),yv.pop(0)])
    convexHull.append([xv.pop(0),yv.pop(0)])
    while len(xv)>0:
        turn = calculateTurn(convexHull[-1], convexHull[-2],[xv[0],yv[0]])
        if turn == t:
            convexHull.append([xv.pop(0),yv.pop(0)])
        else:
            while len(convexHull)>1:
                if alg.intersectionPoint(towerLine,[convexHull[-
2],[xv[0],yv[0]]])[1]>yVerticeStart:
                    criticalPairs.append([convexHull[-2],[xv[0],yv[0]]])

                turn = calculateTurn(convexHull[-1], convexHull[-2],[xv[0],yv[0]])
                if turn == t*-1:
                    convexHull.pop(-1)
```

```
                else:
                    break
            convexHull.append([xv.pop(0),yv.pop(0)])
    return criticalPairs
```

3.3.1.20 Συνάρτηση calculateTurn(v1, v2, v3):

Η συνάρτηση calculateTurn(v1, v2, v3) παίρνει ως όρισμα τρεις κορυφές που αποτελούν δύο διαδοχικές ακμές και επιστρέφει 1 εάν δημιουργούν δεξιά στροφή ή δεν δημιουργούν στροφή και 0 στην άλλη περίπτωση.

```
def calculateTurn(v1, v2, v3):
    result = (v2[0] - v1[0]) * (v3[1] - v1[1]) - (v2[1] - v1[1]) * (v3[0] - v1[0])
    if result >= 0:
        return 1
    else:
        return -1
```

3.3.1.21 Διαδικασία visualizeSemicontinious()

Η διαδικασία visualizeSemicontinious() καλεί τη συνάρτηση calculateSecondTowerSemiContiniousSolo() και κατασκευάζει τη γραφική αναπαράσταση της λύσης της ημι-συνεχούς εκδοχής του προβλήματος.

```
def visualizeSemicontinious():
    towers = calculateSecondTowerSemiContiniousSolo()
    tower1 = towers[0]
    tower2 = towers[1]
    upperEnvelope = towers[2]
    plt.figure(num='The two watchtowers Semi')
    plt.title("Semi-Continious Problem")
    plt.plot(xOfVertices,yOfVertices,color = 'green')
    plt.plot(xOfVertices, yOfVertices, marker = ".", color = "green")
    if len(upperEnvelope)>0:
```

```
        plt.plot(upperEnvelope[0],upperEnvelope[1],color = 'blue')
        plt.plot(upperEnvelope[0], upperEnvelope[1], marker = ".", color = "blue")

    plt.plot([tower1[0],tower1[0]],[tower1[1],tower1[1]+tower1[2]],color = '0')
    plt.plot([tower1[0]],[tower1[1]+tower1[2]], marker = "s",color = "red")
    plt.plot([tower2[0],tower2[0]],[tower2[1],tower2[1]+tower2[2]],color = '0')
    plt.plot([tower2[0]],[tower2[1]+tower2[2]], marker = "s",color = "blue")
    plt.show()
```

3.3.1.22 Διαδικασία visualizeDiscrete()

Η διαδικασία visualizeDiscrete() καλεί τη συνάρτηση calculateSecondTowerDiscreteSolo() και κατασκευάζει τη γραφική αναπαράσταση της λύσης της διακριτής εκδοχής του προβλήματος.

```
def visualizeDiscrete():
    towers = calculateSecondTowerDiscreteSolo()
    tower1 = towers[0]
    tower2 = towers[1]
    index1 = xOfVertices.index(tower1[0])
    index2 = xOfVertices.index(tower2[0])
    tower1.insert(1,yOfVertices[index1])
    tower2.insert(1,yOfVertices[index2])
    plt.figure(num='The two watchtowers Discrete')
    plt.title("Discrete Problem")
    plt.plot(xOfVertices,yOfVertices,color = 'green')
    plt.plot(xOfVertices, yOfVertices, marker = ".", color = "green")
    plt.plot([tower1[0],tower1[0]],[tower1[1],tower1[1]+tower1[2]],color = '0')
    plt.plot([tower1[0]],[tower1[1]+tower1[2]], marker = "s",color = 'red')
    plt.plot([tower2[0],tower2[0]],[tower2[1],tower2[1]+tower2[2]],color = '0')
    plt.plot([tower2[0]],[tower2[1]+tower2[2]], marker = "s",color = 'blue')
    y = min(yOfVertices)
    autoUpdateTower(tower1[0],tower1[1]+tower1[2])
    autoCalculateVisibilityLines()
    for hiddenPiece in setNotVisibleEdges:
```

```
        plt.plot([hiddenPiece[0][0],hiddenPiece[1][0]],[hiddenPiece[0][1],hiddenPiece[1
        ][1]],linestyle = 'dotted', color = 'red')

    for endPoint in setVisP:

        plt.plot([xTower, endPoint[0]],[yTower,endPoint[1]], linestyle =
'dotted',color = 'red')

    autoUpdateTower(tower2[0],tower2[1]+tower2[2])

    autoCalculateVisibilityLines()

    for hiddenPiece in setNotVisibleEdges:


        plt.plot([hiddenPiece[0][0],hiddenPiece[1][0]],[hiddenPiece[0][1],hiddenPiece[1
        ][1]],linestyle = 'dotted', color = 'blue')

    for endPoint in setVisP:

        plt.plot([xTower, endPoint[0]],[yTower,endPoint[1]], linestyle =
'dotted',color = 'blue')

    plt.show()
```

3.3.1.23 Διαδικασία calculateSecondTower()

Η διαδικασία calculateSecondTower() είναι μία από τις ενδιάμεσες διαδικασίες του προβλήματος και υπολογίζει μία λύση του προβλήματος για έναν επιλεγμένο πύργο και την οπτικοποιεί.

```
def calculateSecondTower():

    autoCalculateVisibilityLines()

    visibilityPairsLeft = autoCalculateLeftPUV() #O(n^2logn)

    visibilityPairsRight = autoCalculateRightPUV() #O(n^2logn)

    setEv =
visibilityPairsLeft[0]+visibilityPairsLeft[1]+visibilityPairsRight[0]+visibilityPairsL
eft[1]

    currentTower2 = 0

    currentHeight2 = sys.float_info.max

    for j in range(len(xOfVertices)):


        secondTowerLine =
[[xOfVertices[j],yOfVertices[j]],[xOfVertices[j],yOfVertices[j]+2]]

        possibleHeights2 = []

        for line in setEv:

            if line[0][0] != line[1][0]:
```

```
                temp = alg.intersectionPoint(line ,secondTowerLine)[1]
                if round(temp,10) >= yOfVertices[j]:
                    possibleHeights2.append(round(temp,10))
        if len(possibleHeights2) > 0:
            currentY2 = max(possibleHeights2)
            tempHeight2 = currentY2 - yOfVertices[j]
        else:
            tempHeight2 = sys.float_info.max
        print(xOfVertices[j], tempHeight2 , currentHeight2)
        if tempHeight2 < currentHeight2:
            currentHeight2 = tempHeight2
            currentTower2 = j
    if currentHeight2 == sys.float_info.max:
        currentHeight2 = 0
    drawTerrain()

plt.plot([xOfVertices[currentTower2],xOfVertices[currentTower2]],[yOfVertices[currentT
ower2],yOfVertices[currentTower2]+currentHeight2],color = '0')
    plt.plot([xOfVertices[currentTower2]],[yOfVertices[currentTower2]+currentHeight2],
marker = "s")
    plt.show()
    print(xOfVertices[currentTower2],currentHeight2)
```

3.3.1.24 Διαδικασία visualizeUpperEnvelope()

Η διαδικασία visualizeUpperEnvelope() υπολογίζει και οπτικοποιεί για έναν επιλεγμένο πύργο το upper envelope για όλους τους φορείς των αριστερών και δεξιών ζευγών ορατότητας και εμφανίζει την γραφική του αναπαράσταση.

```
def visualizeUpperEnvelope():
    autoCalculateVisibilityLines() #O(n)
    visibilityPairsLeft = autoCalculateLeftPUV() #O(n^2logn)
    visibilityPairsRight = autoCalculateRightPUV() #O(n^2logn)
    setB = visibilityPairsLeft[0]
    setC = visibilityPairsRight[0]
    setNotVisE = visibilityPairsRight[1]
```

```
    setLines = setB+setC+setNotVisE
    upperEnvelope = calculateUpperEnvelope(setLines)
    plt.plot(upperEnvelope[0],upperEnvelope[1],color = 'blue')
    plt.plot(upperEnvelope[0], upperEnvelope[1], marker = ".", color = "blue")
    plt.show()
```

3.3.1.25 Διαδικασία drawVisibilityLines()

Η διαδικασία drawVisibilityLines() υπολογίζει την ορατότητα ενός επιλεγμένου πύργου και εμφανίζει τη γραφική αναπαράσταση της ορατότητας που καλύπτει ο πύργος στη Τ.

```
def drawVisibilityLines():
    global setP, terrainWithP, setNotVisibleEdges
    setNotVisibleEdges = []


    plt.close()
    drawTerrain()
    endPoints = alg.visibilityEndpoints(xOfVertices, yOfVertices, xTower, yTower)
#returns [[endpoints- (x,y)], [endpoints and vertices out of sight- (x,y)]]
    setPonVertice= []
    for endPoint in endPoints[0]:
        if endPoint[0] in xOfVertices:
            setPonVertice.append(endPoint)


    print(endPoints[2])
    setP = endPoints[0]
    for i in range(len(setP)):
        setP[i][0] = round(setP[i][0],13)
        setP[i][1] = round(setP[i][1],13)
    terrainWithP = [xOfVertices.copy(),yOfVertices.copy()]
    setPwithNoVertice = copy.deepcopy(setP)
    epCounter = 0
    if len(setPwithNoVertice)>0:
        for i in range(len(xOfVertices)):
            if len(setPwithNoVertice)>0:
```

```
                if epCounter >= len(setPwithNoVertice):
                        break
                if  xOfVertices[i] == setPwithNoVertice[epCounter][0]:
                    setPwithNoVertice.remove(setPwithNoVertice[epCounter])
                    continue
                if  xOfVertices[i] > setPwithNoVertice[epCounter][0]:

terrainWithP[0].insert(epCounter+i,setPwithNoVertice[epCounter][0])

terrainWithP[1].insert(epCounter+i,setPwithNoVertice[epCounter][1])
                    epCounter+=1


        for endPoint in setP+setPonVertice:
            plt.plot([endPoint[0]],[endPoint[1]], marker = ".")
            plt.plot([xTower, endPoint[0]],[yTower,endPoint[1]], linestyle =
'dotted',color = '0')



    for p in setPonVertice:
        endPoints[1].remove(p)
    print(endPoints[2])
    for i in range (len(endPoints[1])-1):
        if endPoints[1][i] in endPoints[2] and endPoints[1][i+1]in endPoints[2]:
            continue
        if endPoints[1][i][0] in xOfVertices and endPoints[1][i+1][0] in xOfVertices:
            start = xOfVertices.index(endPoints[1][i][0])
            finish = xOfVertices.index(endPoints[1][i+1][0])
            if abs(start - finish) == 1:
                setNotVisibleEdges.append([endPoints[1][i],endPoints[1][i+1]])
        if endPoints[1][i][0] in xOfVertices and endPoints[1][i+1] in setP and
endPoints[1][i+1] not in setPonVertice:
            setNotVisibleEdges.append([endPoints[1][i],endPoints[1][i+1]])
    setPnotValuable = copy.deepcopy(setPonVertice)


    for point in setPnotValuable:
        counter = 0
        for pair in setNotVisibleEdges:
```

```
            for p in pair:
                if p == point:
                    counter+=1
        if counter==1:
            setPnotValuable.remove(point)
    for p in setP:
        if p in setPnotValuable:
            setP.remove(p)



    for hiddenPiece in setNotVisibleEdges:
        for v in hiddenPiece:
            if v not in setP and v not in endPoints[2]:
                setP.append(v)
    setP = sorted(setP, key=lambda p:p[0])


    for hiddenPiece in setNotVisibleEdges:

plt.plot([hiddenPiece[0][0],hiddenPiece[1][0]],[hiddenPiece[0][1],hiddenPiece[1][1]],
linewidth=2,linestyle = 'dashed', color = '0')
    plt.show()
```

3.3.1.26 Διαδικασία calculateLeftPUV()

Η διαδικασία calculateLeftPUV() υπολογίζει τα αριστερά ζεύγη ορατότητας για έναν επιλεγμένο πύργο και εμφανίζει την γραφική τους αναπαράσταση.

```
def calculateLeftPUV():
    setE = []
    puv = dc.constructPUV(xOfVertices, terrainWithP,setP)
    for part in setNotVisibleEdges:
        setE.append(part)
    leftPV = [puv,setE]
    for pair in leftPV[0]:
        copyPair = copy.deepcopy(pair)


        rePair = [copyPair[1],copyPair[0]]
```

```
        if pair[1][0] not in xOfVertices and  rePair in leftPV[0]:
            leftPV[0].remove(pair)


    drawTerrain()
    for pair in leftPV[0]+leftPV[1]:
        plt.plot([pair[0][0], pair[1][0]],[pair[0][1],pair[1][1]], linestyle =
'dotted',color = '0')


    plt.show()
```

3.3.1.27 Διαδικασία calculateRightPUV()

Η διαδικασία calculateRightPUV() υπολογίζει τα δεξιά ζεύγη ορατότητας για έναν επιλεγμένο πύργο και εμφανίζει την γραφική τους αναπαράσταση.

```
def calculateRightPUV():
    setE = []
    rightTerrain = copy.deepcopy(terrainWithP)
    rightSetP = copy.deepcopy(setP)
    rightTerrain[0] = [rightTerrain[0][-1] - x for x in rightTerrain[0]]


    for i in range(len(rightSetP)):
        rightSetP[i][0] = terrainWithP[0][-1] - rightSetP[i][0]


    rightTerrain[0].reverse()
    rightTerrain[1].reverse()
    rightSetP.reverse()


    revXofVertices = copy.deepcopy(xOfVertices)
    revXofVertices = [xOfVertices[-1] - x for x in xOfVertices]
    rightPUV = dc.constructPUV(revXofVertices, rightTerrain,rightSetP)
    print(rightSetP, rightPUV)
    plt.figure()
    for i in range(len(rightPUV)):
        rightPUV[i][0][0] = xOfVertices[-1]-rightPUV[i][0][0]
```

```
        rightPUV[i][1][0] = xOfVertices[-1]-rightPUV[i][1][0]
    for part in setNotVisibleEdges:
        setE.append(part)
    rightPV = [rightPUV,setE]
    plt.plot(xOfVertices,yOfVertices,color = 'green')
    plt.plot(xOfVertices, yOfVertices, marker = ".", color = "green")
    if numberTower != -1:
        plt.plot([xTower,xTower],[yBase,yTower],color = '0')
        plt.plot([xTower],[yTower], marker = "s")
    for pair in rightPV[0]:
        plt.plot([pair[0][0], pair[1][0]],[pair[0][1],pair[1][1]], linestyle =
'dotted',color = '0')


    plt.show()
```

3.3.2 Αρχείο algorithm.py

Στο αρχείο algorithm.py κατασκευάσαμε χρήσιμες βοηθητικές συναρτήσεις που χρησιμοποιούμε πολλαπλές φορές σε όλα τα αρχεία για να πετύχουμε τις ανάγκες του αλγορίθμου.

3.3.2.1 Συνάστηση visibilityEndpoints(xOfVertices, yOfVertices, xTower, yTower)

Η συνάρτηση visibilityEndpoints(xOfVertices, yOfVertices, xTower, yTower) παίρνει ως είσοδο τις συντεταγμένες των κορυφών της T και το άνω άκρο ενός πύργου και υπολογίζει και επιστρέφει τα σημεία των μερικώς μη ορατών ακμών που χάνεται η ορατότητα, τις μη ορατές κορυφές και τις οριακά ορατές κορυφές.

```
#gets the terrain and the tower and returns the point that are not visible (Endpoints
of part visible edges and vertexes) O(n) time

def visibilityEndpoints(xOfVertices, yOfVertices, xTower, yTower):
    numberTower = xOfVertices.index(xTower)
    curSlop = 0
    minSlop = sys.float_info.max
    segment = []
    ray = []
    endPoints = []
    outOfSight = []
    inSight= []
    #calculate endPoints left of the first tower
    for i in range(numberTower-1,-1,-1):
    #because terrain is x monotonous I use the slope for finding the shady parts of
the terrain
    # If a slope is lesser than its previous then there are unwatched edges or part of
edge
        curSlop = (yTower-yOfVertices[i])/(xTower-xOfVertices[i])
        if curSlop<=minSlop:
            minSlop = curSlop
            if i < numberTower-1:
                segment =
[[xOfVertices[i],yOfVertices[i]],[xOfVertices[i+1],yOfVertices[i+1]]]
                endPoint = intersectionPoint(ray,segment)
                if endPoint == 0: #critically visible edge
                    inSight.append([xOfVertices[i],yOfVertices[i]])
                    continue
                endPoint[0] = round(endPoint[0],13)
                endPoint[1] = round(endPoint[1],13)
                if ray[0]!=segment[1]:
                    endPoints.insert(0,endPoint)
                    outOfSight.append(endPoint)
            ray = [[xOfVertices[i],yOfVertices[i]],[xTower,yTower]]
            if i>0:
                nextSlop = (yTower-yOfVertices[i-1])/(xTower-xOfVertices[i-1])
                if curSlop < nextSlop:
                    outOfSight.append([xOfVertices[i],yOfVertices[i]])
                    inSight.append([xOfVertices[i],yOfVertices[i]])
        else:
            if xOfVertices[i+1] not in outOfSight:
                outOfSight.append([xOfVertices[i+1],yOfVertices[i+1]])
            outOfSight.append([xOfVertices[i],yOfVertices[i]])

    curSlop = 0
    maxSlop = sys.float_info.max*-1
    segment = []
    ray = []
    rightOutOfSight = []

    #calculate endPoints right of the first tower
    for i in range(numberTower+1,len(xOfVertices)):
        curSlop = (yTower-yOfVertices[i])/(xTower-xOfVertices[i])
```

```
    #because terrain is x monotonous I use the slope for finding the shady parts of
the terrain
    #If a slope is lesser than its previous then there are unwatched edges or part of
edge
        if curSlop>=maxSlop:
            maxSlop = curSlop
            if i > numberTower+1:

                segment = [[xOfVertices[i],yOfVertices[i]],[xOfVertices[i-
1],yOfVertices[i-1]]]
                endPoint = intersectionPoint(ray,segment)
                if endPoint == 0: #critical visible vertice
                    inSight.append([xOfVertices[i],yOfVertices[i]])
                    continue
                endPoint[0] = round(endPoint[0],13)
                endPoint[1] = round(endPoint[1],13)

                if ray[0]!=segment[1]:
                    if xOfVertices[i-1] not in outOfSight:
                        outOfSight.append([xOfVertices[i-1],yOfVertices[i-1]])
                    endPoints.append(endPoint)
                    outOfSight.append(endPoint)
                    rightOutOfSight.append(endPoint)
                    inSight.append(endPoint)
            ray = [[xOfVertices[i],yOfVertices[i]],[xTower,yTower]]
            if i<len(xOfVertices)-1:
                nextSlop = (yTower-yOfVertices[i+1])/(xTower-xOfVertices[i+1])
                if curSlop > nextSlop:
                    outOfSight.append([xOfVertices[i],yOfVertices[i]])
                    inSight.append([xOfVertices[i],yOfVertices[i]])
                    rightOutOfSight.append([xOfVertices[i],yOfVertices[i]])
        else:
            outOfSight.append([xOfVertices[i],yOfVertices[i]])

            rightOutOfSight.append([xOfVertices[i],yOfVertices[i]])

    return [endPoints, outOfSight,inSight]
```

3.3.2.2 Συνάρτηση intersectionPoint(line1, line2)

Η συνάρτηση intersectionPoint(line1, line2) παίρνει ως όρισμα δύο ευθύγραμμα τμήματα και υπολογίζει το σημείο τομής τους και το επιστρέφει. Αν δεν υπάρχει σημείο τομής επιστρέφει τον αριθμό μηδέν.

```
#gets 2 lines and returns the intersection

def intersectionPoint(line1, line2): #line1 =  [[x1line1,y1line1],[x2line1,y2line1]],
line2 same

    xdiff = [line1[0][0] - line1[1][0], line2[0][0] - line2[1][0]]
```

```
    ydiff = [line1[0][1] - line1[1][1], line2[0][1] - line2[1][1]]

    def det(a, b):
        return a[0] * b[1] - a[1] * b[0]

    div = det(xdiff, ydiff)
    if div == 0:
       return 0

    d = [det(*line1), det(*line2)]
    x = det(d, xdiff) / div
    y = det(d, ydiff) / div
    return [x, y]
```

3.3.2.3 Συνάρτηση pairPV(xOfVertices, subTerrain, setP, setPV)

Η συνάρτηση pairPV(xOfVertices, subTerrain, setP, setPV) παίρνει ως ορίσματα τις τετμημένες των κορυφών της T, ένα τμήμα της T που εμπεριέχει τα σημεία P, το σύνολο P και μία κενή λίστα. Με αυτή τη συνάρτηση υπολογίζουμε τα αριστερά ζεύγη ορατότητας τα οποία προσαρτούμε στην αρχικά κενή λίστα και τέλος επιστρέφουμε την λίστα με όλα τα αριστερά ζεύγη ορατότητας.

```
#recursevly calculating for rightmost vertex which p endpoint can see appends it to
the set and calls the next vertex until PLVL is empty
def pairPV(xOfVertices, subTerrain, setP, setPV):
    if len(subTerrain[0]) == 0:
        return setPV
    Vx = subTerrain[0].pop(-1)
    Vy = subTerrain[1].pop(-1)
    while Vx not in xOfVertices and len(subTerrain[0])>0:
        Vx = subTerrain[0].pop(-1)
        Vy = subTerrain[1].pop(-1)
    minSlop = sys.float_info.max
    pair = []
    if len(subTerrain[0]) == 0:
        return setPV
```

```
    else:
        for i in range(len(subTerrain[0])):
            curSlop = (Vy-subTerrain[1][-i-1])/(Vx-subTerrain[0][-i-1])
            if curSlop<minSlop:
                minSlop = curSlop
                if [subTerrain[0][-i-1],subTerrain[1][-i-1]] in setP:
                    pair = [[subTerrain[0][-i-1],subTerrain[1][-i-1]],[Vx,Vy]]
                    setPV.append(pair)
                    break
        return pairPV(xOfVertices, subTerrain, setP, setPV)
```

3.3.2.4 Συνάρτηση calculateVC(vR,notVR)

Η συνάρτηση calculateVC(vR,notVR) παίρνει ως όρισμα ένα σύνολο κορυφών vR και κάποιες από αυτές τις κορυφές notVR και υπολογίζει και επιστρέφει την κορυφή που επιτρέπει σε μία κορυφή από το notVR να να «δει» στο χαμηλότερο δυνατό σημείο τον κατακόρυφο φορέα της αριστερότερης κορυφής του vR.

```
#inputs: vertices right of l vertical line and VR' vertices
#for every VR' vertex calculate vertex that allows VR' to see l vertical line
def calculateVC(vR,notVR):
    c=[[],[]]
    for i in range(len(vR[0])-1,-1,-1):
        if vR[0][i] in notVR[0]:
            minSlope = (vR[1][i]-vR[1][i-1])/(vR[0][i]-vR[0][i-1])
            currC = [vR[0][i-1],vR[1][i-1]]
            for j in range(i-1,-1,-1):
                curSlope = (vR[1][j]-vR[1][i])/(vR[0][j]-vR[0][i])
                if curSlope < minSlope:
                    minSlope = curSlope
                    currC = [vR[0][j],vR[1][j]]
            c[0].insert(0,currC[0])
            c[1].insert(0,currC[1])
    return c
```

3.3.2.5 Συνάρτηση calculatePA(pL,vL)

Η συνάρτηση calculatePA(pL,vL) παίρνει ως όρισμα ένα σύνολο σημείων P, pL, και ένα σύνολο κορυφών και σημείων P, vL, τμήματος της T και υπολογίζει και επιστρέφει την κορυφή που επιτρέπει σε ένα σημείο από το pL να «δει» στο χαμηλότερο δυνατό σημείο τον κατακόρυφο φορέα της δεξιότερης κορυφής του vL.

```
def calculatePA(pL,vL):
    a = [[],[]]
    for i in range(len(pL[0])):
        while 1:


            if len(vL[0])==0:
                return a
            if vL[0][0]<= pL[0][i]:
                vL[0].pop(0)
                vL[1].pop(0)
            else:
                break
        maxSlop = (vL[1][0]-pL[1][i])/(vL[0][0]-pL[0][i])
        currA = [vL[0][0], vL[1][0]]
        for j in range(1,len(vL[0])):
            curSlop = (vL[1][j]-pL[1][i])/(vL[0][j]-pL[0][i])
            if curSlop > maxSlop:
                maxSlop = curSlop
                currA = [vL[0][j], vL[1][j]]


        a[0].append(currA[0])
        a[1].append(currA[1])
    return a
```

3.3.2.5 Συνάρτηση calculateLineFrom2Points(a,b)

Η συνάρτηση calculateLineFrom2Points(a,b) παίρνει ως όρισμα δύο σημεία στο επίπεδο και επιστρέφει την κλίση και την σταθερά της εξίσωσης της ευθείας που σχηματίζουν.

```
def calculateLineFrom2Points(a,b): #a = [ax,ay] and b = [bx,by]
    wpm = (a[1] - b[1])/(a[0]-b[0])
    wpn = a[1] - wpm * a[0]
    return [wpm,wpn]
```

3.3.3 Αρχείο divideAndConquer.py

Στο αρχείο divideAndConquer.py υλοποιήσαμε την αναδρομική διαδικασία που παράγει τα αριστερά και δεξιά ζευγάρια ορατότητας καθώς και τα οπτικοποιεί σε μεμονωμένα παραδείγματα για κάποιο συγκεκριμένο πύργο.

3.3.3.1 Συνάρτηση constructPUV(xOfVertices,terrain,setP)

Η συνάρτηση constructPUV(xOfVertices,terrain,setP) υλοποιεί την διαίρει και βασίλευε διαδικασία και υπολογίζει και επιστρέφει τα αριστερά ζεύγη ορατότητας για τη T και τα σημεία P που παίρνει ως όρισματα.

```
def constructPUV(xOfVertices,terrain,setP): #terrain =
[[x1,x2,x3,...,xn],[y1,y2,y3,...,yn]]
    middle = int(len(terrain[0])/2)
    leftTerrain = [[],[]]
    rightTerrain = [[],[]]
    l = []
    r = []
    pL = [[],[]]
    plotInfo = []
    for i in range(len(terrain[0])):
        if middle >= i:
            leftTerrain[0].append(terrain[0][i])
            leftTerrain[1].append(terrain[1][i])
        if middle <= i:
            rightTerrain[0].append(terrain[0][i])
            rightTerrain[1].append(terrain[1][i])
```

```
if len(leftTerrain[0])>6:
    l = constructPUV(xOfVertices, leftTerrain,setP)

else:
    l = alg.pairPV(xOfVertices,copy.deepcopy(leftTerrain), setP, [])
if len(rightTerrain[0])>6:
    r = constructPUV(xOfVertices, rightTerrain,setP)
else:
    r = alg.pairPV(xOfVertices,copy.deepcopy(rightTerrain), setP, [])



vL = copy.deepcopy(leftTerrain)
notPairPRVR = copy.deepcopy(rightTerrain)
if notPairPRVR[0][0] == rightTerrain[0][0]:
    notPairPRVR[0].pop(0)
    notPairPRVR[1].pop(0)
tempTerrain = copy.deepcopy(terrain) #current terrain without p points
for pair in r:
    index = notPairPRVR[0].index(pair[1][0])
    notPairPRVR[0].pop(index)
    notPairPRVR[1].pop(index)
for p in setP:
    if p[0] in notPairPRVR[0]:
        index = notPairPRVR[0].index(p[0])
        notPairPRVR[0].pop(index)
        notPairPRVR[1].pop(index)
    if p[0]>=leftTerrain[0][0] and p[0]<= leftTerrain[0][-1]:
        pL[0].append(p[0])
        pL[1].append(p[1])



if len(pL[0])>0 and len(notPairPRVR[0])>0 :
    #a einai h koryfh v poy epitrepei sto pL shmeio na dei thn katakoryfh l
    a = alg.calculatePA(copy.deepcopy(pL),copy.deepcopy(vL)) ###########kenh lista

    #c einai to antistoixo a omws gia tis notPairPRVR koryfes
```

```
        c = alg.calculateVC(copy.deepcopy(rightTerrain),copy.deepcopy(notPairPRVR))
        interPA = [[],[]] #to shmeio pou temnei h eu8eia tou eu8ugrammou tmhmatos pL-a
sthn l
        interVC = [[],[]] #to shmeio pou temnei h eu8eia tou eu8ugrammou tmhmatos VR'-
c sthn l
        if len(a[0])>0 and len(c[0])>0:
            for i in range (len(a[0])):
                l1 = [[pL[0][i], pL[1][i]],[a[0][i],a[1][i]]]
                l2 = [[rightTerrain[0][0], 0], [rightTerrain[0][0] ,10]]
                temp = alg.intersectionPoint(l1,l2)

                interPA[0].append(temp[0])
                interPA[1].append(temp[1])
            for j in range (len(c[0])):
                l1 = [[notPairPRVR[0][j],notPairPRVR[1][j]],[c[0][j],c[1][j]]]
                l2 = [[rightTerrain[0][0], 0], [rightTerrain[0][0] ,10]]
                temp = alg.intersectionPoint(l1,l2)
                interVC[0].append(temp[0])
                interVC[1].append(temp[1])
            interPA[0] = [x-rightTerrain[0][0] for x in interPA[0]]
            interVC[0] = [x-rightTerrain[0][0] for x in interVC[0]]
            pL[0] = [x-rightTerrain[0][0] for x in pL[0]]
            notPairPRVR[0] = [x-rightTerrain[0][0] for x in notPairPRVR[0]]

            wp = [[],[],[],[]]
            zq = [[],[],[],[]]
            for i in range (len(interPA[0])):
                mnwp =
alg.calculateLineFrom2Points([interPA[0][i],interPA[1][i]],[pL[0][i],pL[1][i]])
                wp[0].append(mnwp[0])
                wp[1].append(mnwp[1])
                #ray gp is y = -pL[0][i]*x + pL[1][i] for x >= m
                #starting point of wp = mnwp

            for i in range (len(interVC[0])):
                if interVC[0][i]!=notPairPRVR[0][i]:
```

```
                    mnzq =
alg.calculateLineFrom2Points([interVC[0][i],interVC[1][i]],[notPairPRVR[0][i],notPairP
RVR[1][i]])
                    zq[0].append(mnzq[0])
                    zq[1].append(mnzq[1])


                else:
                    continue
                    #here vertex is on l so slope is infinite
            #calculate the infinity x of wp and zq


            wpMinX = min(wp[0])-1.0012
            zqMaxX = max(zq[0])+1.0012
            for i in range (len(interPA[0])):
                endX = zqMaxX
                endY = -pL[0][i]*endX + pL[1][i]
                wp[2].append(endX)
                wp[3].append(endY)
            for i in range (len(interVC[0])):
                endX = wpMinX
                endY = -notPairPRVR[0][i]*endX + notPairPRVR[1][i]
                zq[2].append(endX)
                zq[3].append(endY)


            pairs = []


            for i in range (len(zq[0])):
                pairPLVR =[-1,-1]
                mincrossy = sys.float_info.max


                for j in range (len(wp[0])):
                    l1 = [[zq[0][i],zq[1][i]],[zq[2][i],zq[3][i]]]
                    l2 = [[wp[0][j],wp[1][j]],[wp[2][j],wp[3][j]]]


                    cross = alg.intersectionPoint(l1,l2)
```

```
                    if cross[0]>= wp[0][j] and cross[0]<= zq[0][i]:
                        if abs(cross[1]-zq[1][i])<mincrossy:
                            mincrossy = abs(cross[1]-zq[1][i])
                            pairPLVR = [j,i]

                if pairPLVR[0] != -1:

pairs.append([[pL[0][pairPLVR[0]]+rightTerrain[0][0],pL[1][pairPLVR[0]]],[notPairPRVR[
0][pairPLVR[1]]+rightTerrain[0][0],notPairPRVR[1][pairPLVR[1]]]])
            interPA[0] = [x+rightTerrain[0][0] for x in interPA[0]]
            interVC[0] = [x+rightTerrain[0][0] for x in interVC[0]]
            pL[0] = [x+rightTerrain[0][0] for x in pL[0]]
            notPairPRVR[0] = [x+rightTerrain[0][0] for x in notPairPRVR[0]]
            plotInfo.extend([terrain,interPA,interVC,pL,notPairPRVR,wp,zq])
            #plotDNQ(plotInfo)
            return l+r+pairs

    return l+r
```

3.3.3.2 Διαδικασία plotDNQ(plotInfo)

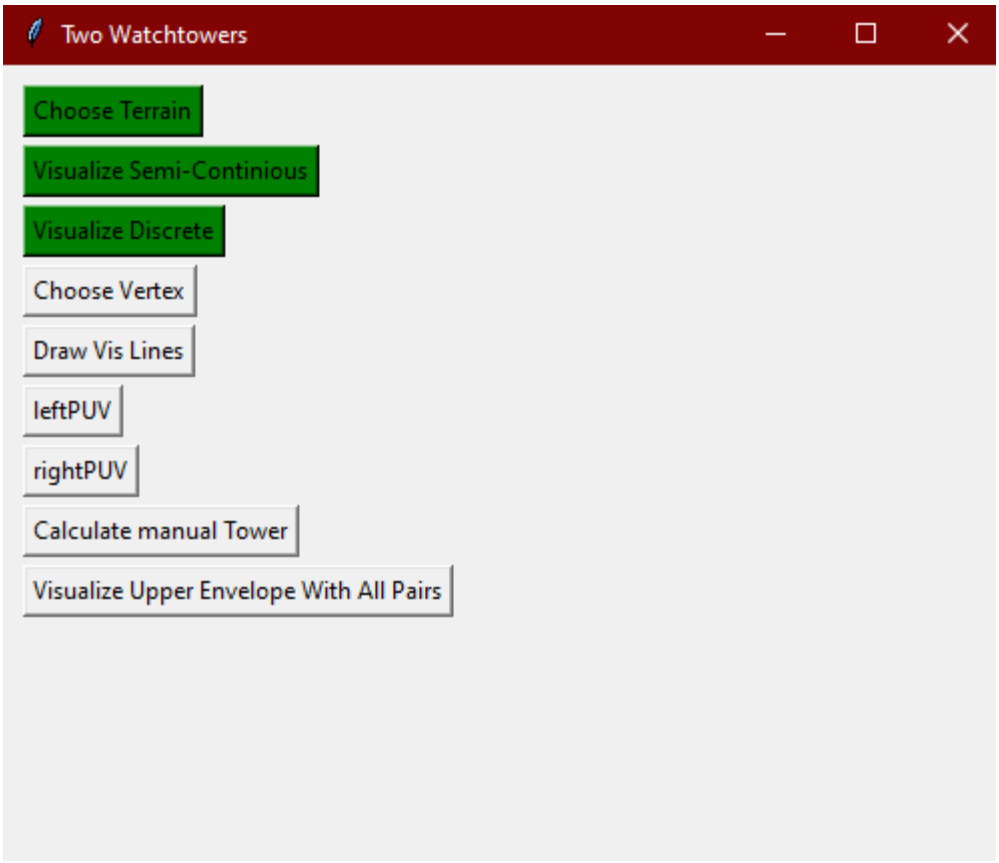
Η διαδικασία plotDNQ(plotInfo) όταν ενεργοποιηθεί από τον προγραμματιστή εμφανίζει την γραφική αναπαράσταση των αριστερών ή δεξιών ζευγών ορατότητας σύμφωνα με τα δεδομένα που δέχεται ως όρισμα.

```
def plotDNQ(plotInfo):
    #terrain[0],interPA[1],interVC[2],pL[3],notPairPRVR[4],wp[5],zq[6]
    #plotInfo[i][0]:x,plotInfo[i][1]:y
    #wp[0]:startx wp[1]:starty wp[2]:endx wp[3]:endy
    #zq ''
    plt.figure()
    plt.subplot(211)
    plt.plot(plotInfo[0][0],plotInfo[0][1], color = "green")
    maxYmid = max([max(plotInfo[1][1]),max(plotInfo[2][1]),max(plotInfo[0][1])])
    minYmid = min([min(plotInfo[1][1]),min(plotInfo[2][1]),min(plotInfo[0][1])])
```

```
    plt.plot([plotInfo[2][0][0], plotInfo[2][0][0]],[maxYmid, minYmid],linestyle =
'dotted',color = 'pink')

    for i in range (len(plotInfo[1][0])):

        plt.plot([plotInfo[1][0][i]], plotInfo[1][1][i], marker = ".", color = "red")

        plt.plot(plotInfo[3][0][i], plotInfo[3][1][i], marker = ".", color = "red")

        plt.plot([plotInfo[3][0][i],plotInfo[1][0][i]],[plotInfo[3][1][i],
plotInfo[1][1][i]],linestyle = 'dotted',color = '0')

    for j in range (len(plotInfo[2][0])):

        plt.plot([plotInfo[2][0][j]], [plotInfo[2][1][j]], marker = ".", color =
"blue")

        plt.plot(plotInfo[4][0][j], plotInfo[4][1][j], marker = ".", color = "blue")

        plt.plot([plotInfo[4][0][j],plotInfo[2][0][j]],[plotInfo[4][1][j],
plotInfo[2][1][j]],linestyle = 'dotted',color = '0')

    plt.subplot(212)

    for i in range (len(plotInfo[5])):

        plt.plot([plotInfo[5][0]], [plotInfo[5][1]], marker = ".", color = "red")


plt.plot([plotInfo[5][0],plotInfo[5][2]],[plotInfo[5][1],plotInfo[5][3]],linestyle =
'dotted',color = 'red')

    for i in range (len(plotInfo[6])):

        plt.plot([plotInfo[6][0]], [plotInfo[6][1]], marker = ".", color = "blue")


plt.plot([plotInfo[6][0],plotInfo[6][2]],[plotInfo[6][1],plotInfo[6][3]],linestyle =
'dotted',color = 'blue')

    plt.show()
```

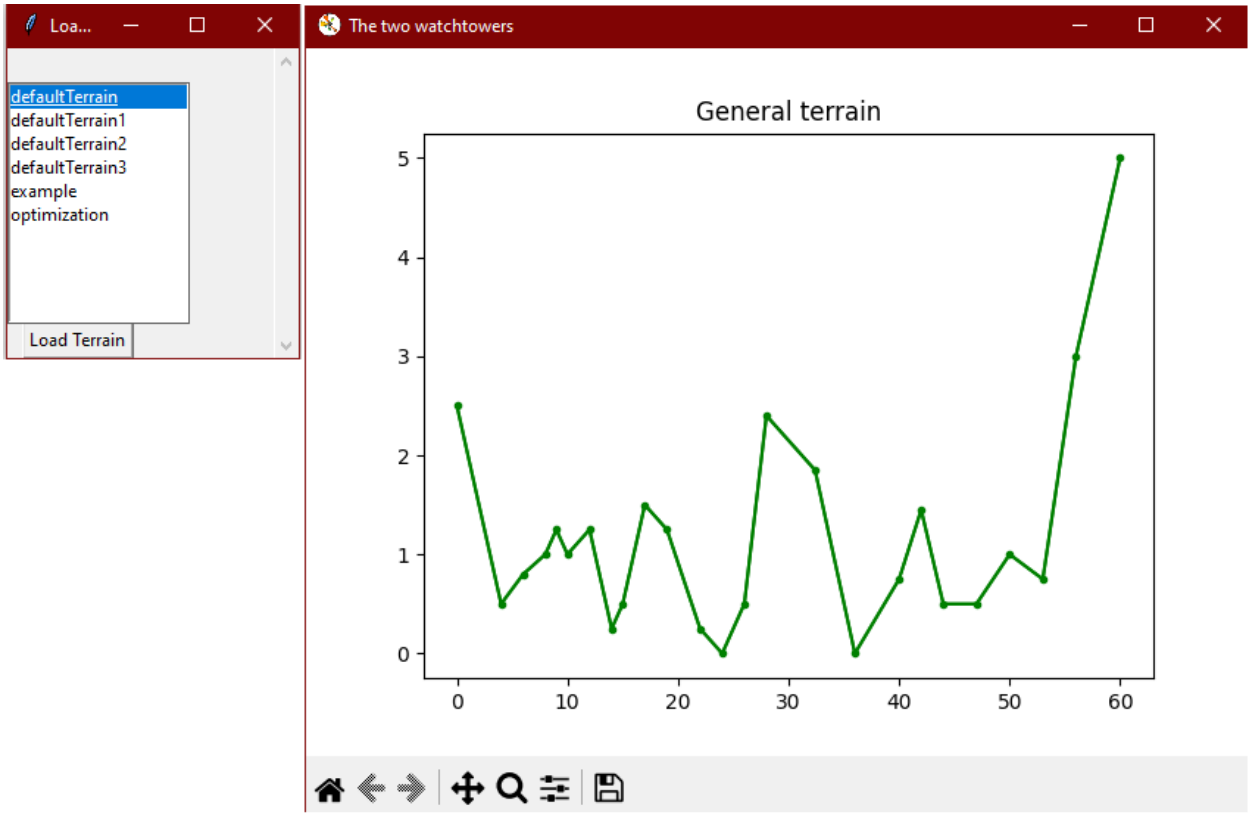
3.4 Παραδείγματα Εκτέλεσης του Προγράμματος

Για να εκκινήσουμε το πρόγραμμα εκτελούμε το αρχείο visualization.py. Παρατηρούμε ότι εμφανίζεται ένα παράθυρο με εννέα κουμπιά. Τρία πράσινα με τις βασικές λειτουργίες και έξι λευκά με τις δευτερεύοντες-ενδιάμεσες διαδικασίες (Σχήμα 3.3).



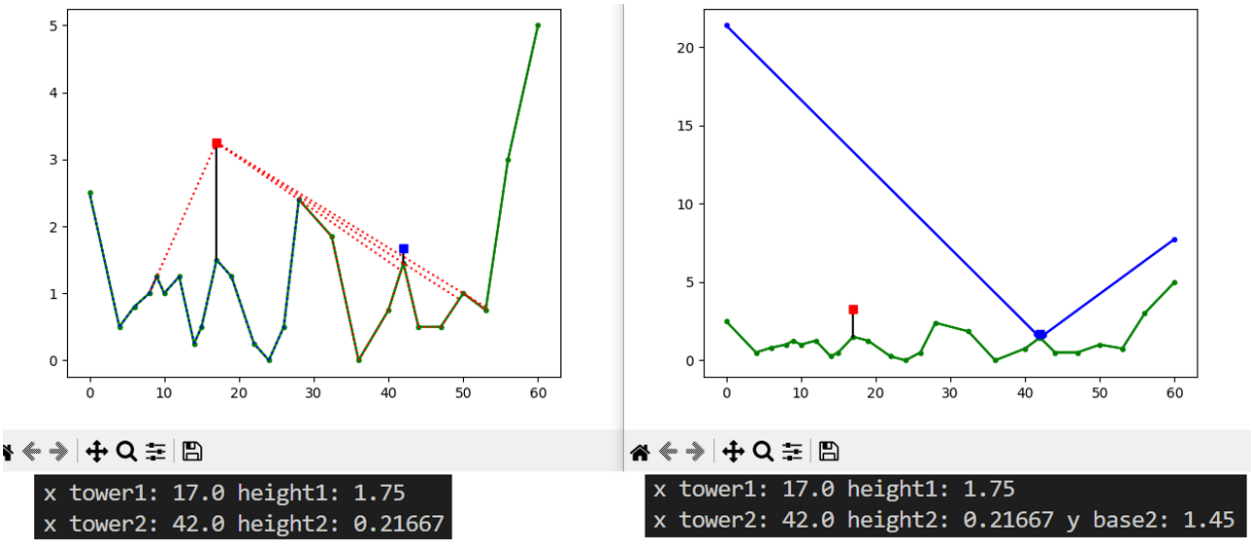
Σχήμα 3.3 *Το παράθυρο που εμφανίζεται κατά την εκτέλεση του visualization.py.*

Το κουμπί «Choose Terrain» ανοίγει ένα παράθυρο που ο χρήστης μπορεί να επιλέξει κάποια από τις Τ που είναι αποθηκευμένες στον φάκελο «terrains». Μόλις επιλέξει μία από τις Τ και πατήσει το κουμπί «Load Terrain» (Σχήμα 3.4) εξαφανίζεται το παράθυρο με τις Τ του φακέλου «terrains» και εμφανίζεται η Τ οπτικοποιημένη σε ένα νέο παράθυρο.

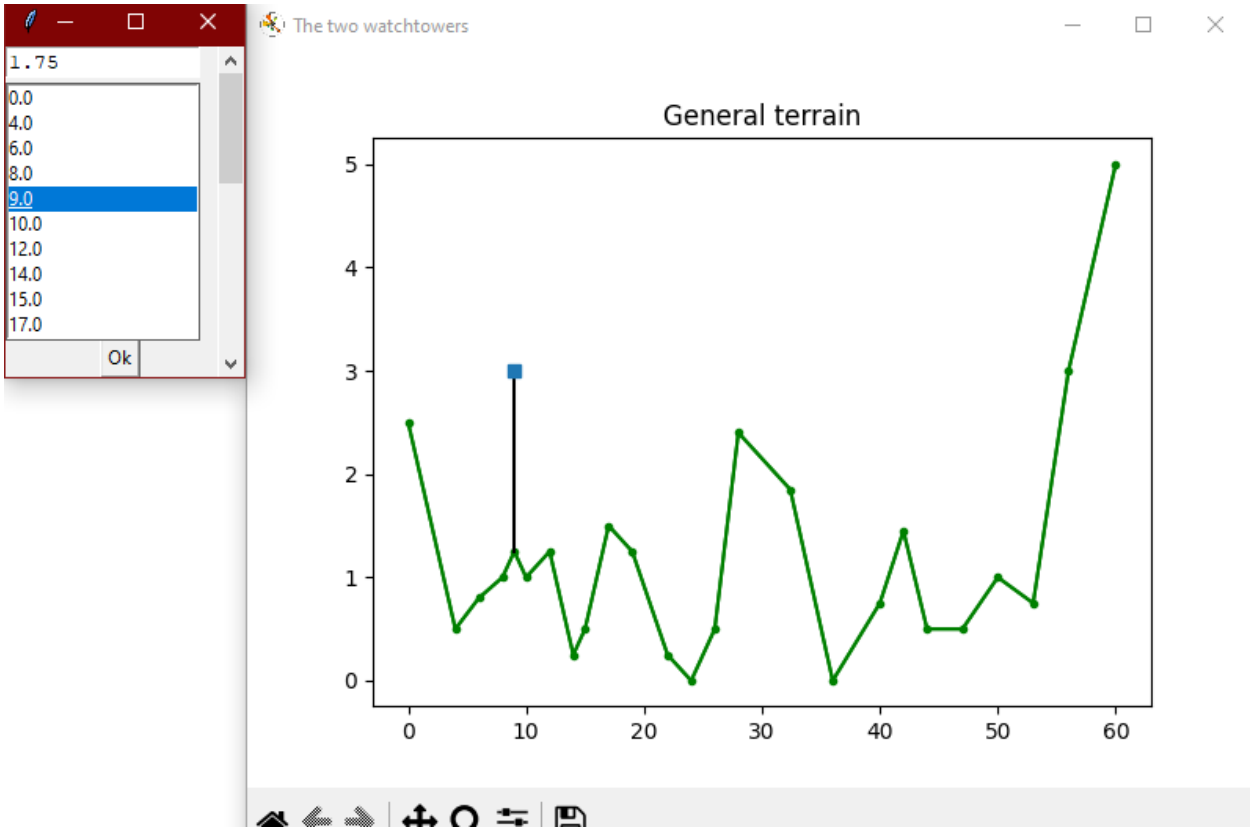


Σχήμα 3.4 *Το παράθυρο με τις Τ αριστερά και το παράθυρο που εμφανίζεται μετά την επιλογή μίας Τ.*

Αφού επιλέξουμε μία Τ μπορούμε να επιλέξουμε είτε το δεύτερο κουμπί «Visualize Semi-Continious», είτε το τρίτο «Visualize Discrete», είτε το τέταρτο «Choose Vertex». Το δεύτερο και το τρίτο δίνουν τις λύσεις για την ημι-συνεχή και διακριτή εκδοχή του προβλήματος αντίστοιχα (Σχήμα 3.5). Όταν επιλέξουμε το κουμπί «Choose Vertex» εμφανίζεται ένα νέο παράθυρο που δίνει στον χρήστη τη δυνατότητα να επιλέξει ένα πρώτο πύργο σε κάποια κορυφή της Τ και με κάποιο ύψος που θα πληκτρολογήσει στο πεδίο «Insert Height». Όταν επιλέξει το κουμπί «Ok» εμφανίζεται ο επιλεγμένος πύργος στο παράθυρο με τη Τ (Σχήμα 3.6). Οι υπόλοιπες επιλογές έχουν νόημα μόνο αν επιλεχθεί κάποιος πύργος από το «Choose Vertex».

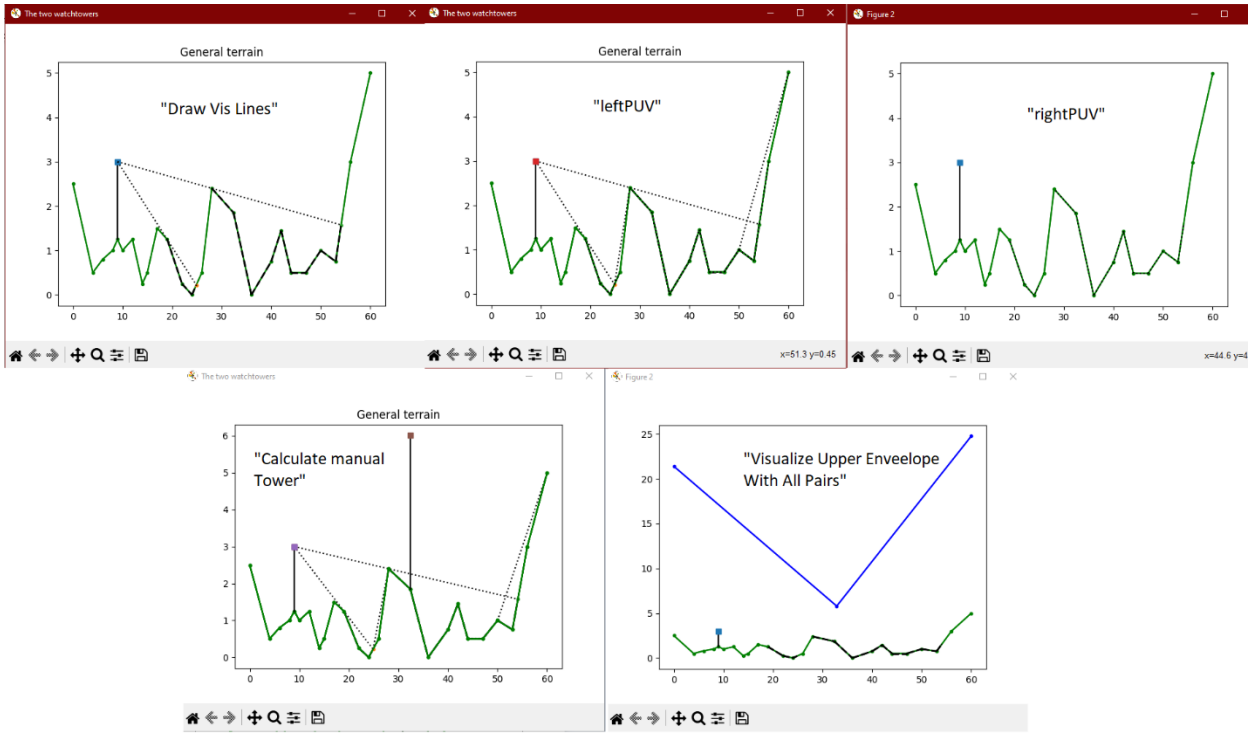


Σχήμα 3.5 *Αριστερά η λύση της διακριτής εκδοχής του προβλήματος για την T του σχήματος 3.4 και κάτω αριστερά η έξοδος στο τερματικό οι τετμημένες των δύο πύργων και το ύψος τους. Δεξιά η λύση της ημι-συνεχούς εκδοχής και η αντίστοιχη έξοδος του τερματικού.*



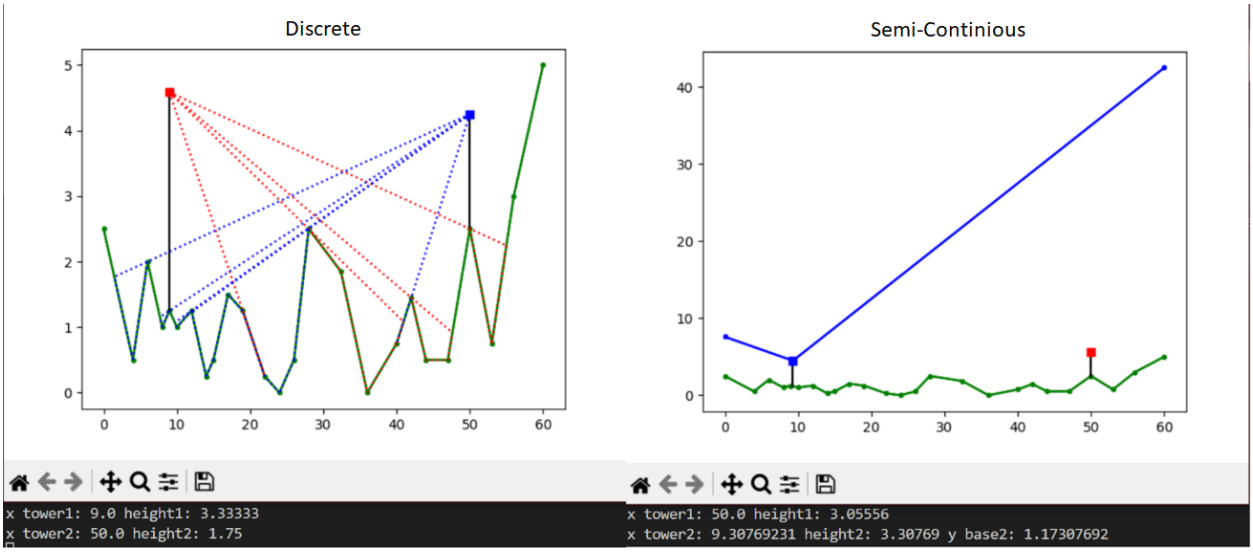
Σχήμα 3.6 *Η επιλογή ενός πύργου στην T με το κουμπί «Choose Vertex».*

Το κουμπί «Draw Vis Lines» εμφανίζει τα σκοτεινά σημεία της T μετά τον ορισμό του πρώτου πύργου w_1 από το «Choose Vertex». Το κουμπί «leftPUV» εμφανίζει τα αριστερά ζεύγη ορατότητας του w_1. Το κουμπί «rightPUV» εμφανίζει τα δεξιά ζεύγη ορατότητας του w_1. Το κουμπί «Calculate manual Tower» εμφανίζει τον δεύτερο κοντύτερο πύργο που επιτηρεί μαζί με τον w_1 την T χωρίς μεγάλη ακρίβεια γιατί χρησιμοποιεί όλους τους φορείς από τα ζεύγη ορατότητας. Το κουμπί «Visualize Upper Envelope With All Pairs» εμφανίζει το upper envelope που έχει κατασκευαστεί από όλους τους φορείς από τα ζεύγη ορατότητας. Στο Σχήμα 3.7 φαίνονται οι λειτουργίες των παραπάνω κουμπιών.

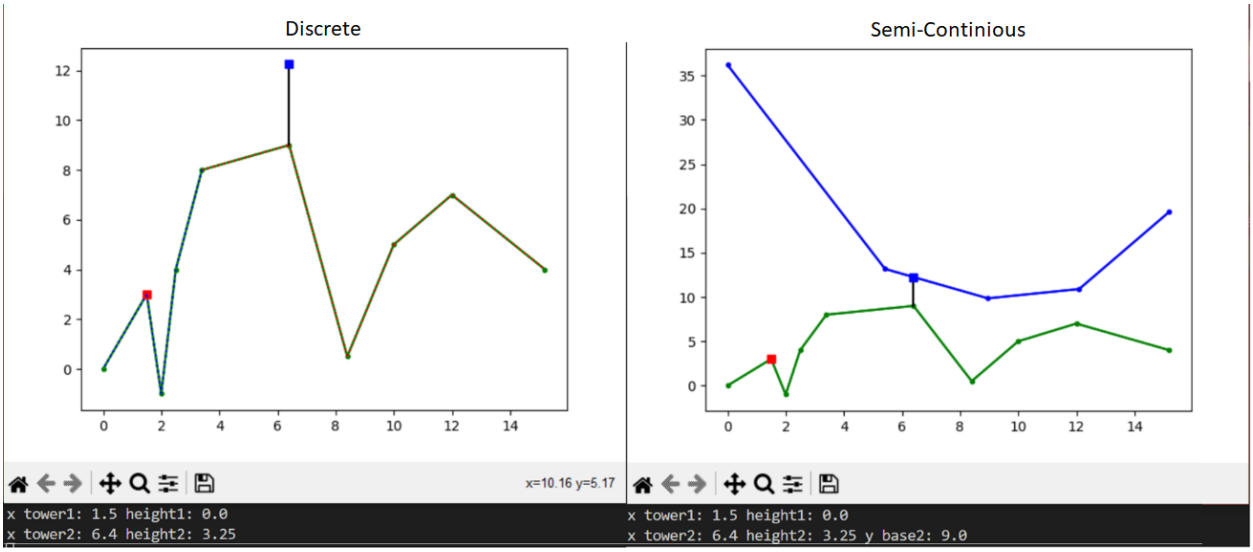


Σχήμα 3.7 *Τα παράθυρα που εμφανίζει το πρόγραμμα όταν πατηθούν τα κουμπιά «leftPUV», «rightPUV», «Calculate manual Tower» και «Visualize Upper Envelope With All Pairs».*

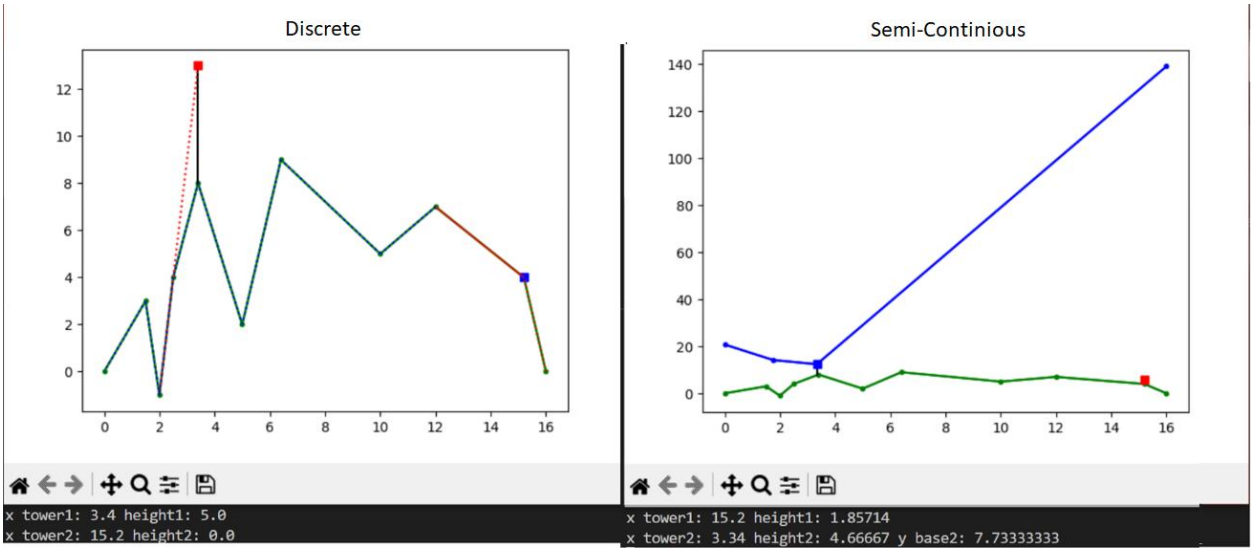
Στα παρακάτω σχήματα παρουσιάζονται οι λύσεις των T (εκτός της πρώτης) που είναι στον φάκελο terrains με τη σειρά του σχήματος 3.4.



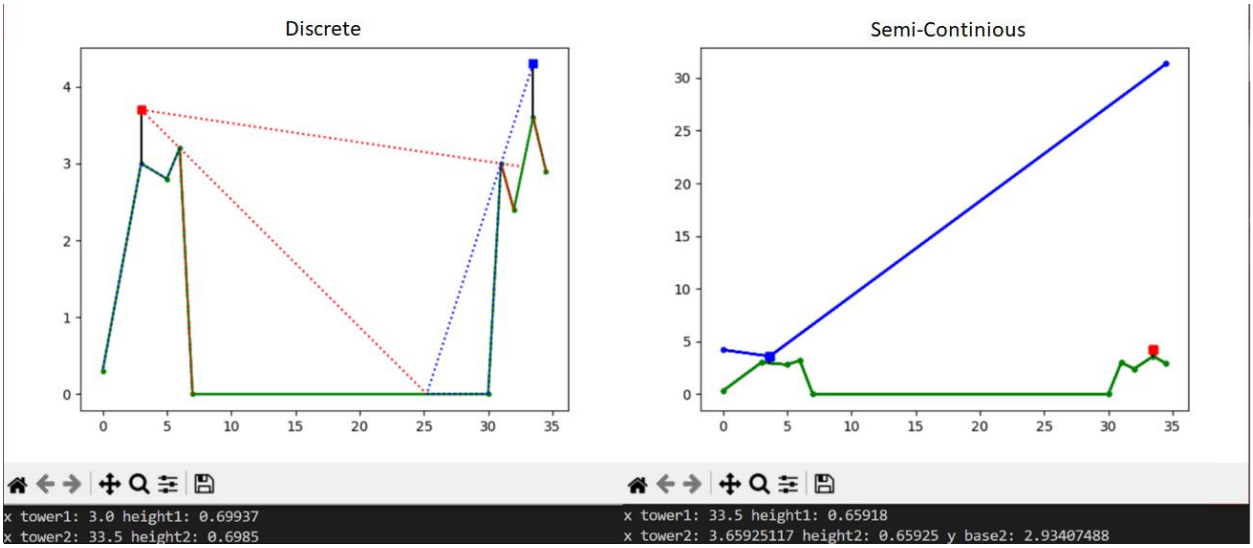
Σχήμα 3.8 *Οι λύσεις για τις δύο εκδοχές της T defaultTerrain1. Για τη διακριτή h=3.33 ενώ για την ημι-συνεχή h=3.3.*



Σχήμα 3.9 *Οι λύσεις για τις δύο εκδοχές της T defaultTerrain2. Για τη διακριτή h=3.25 όπως και την ημι-συνεχή h=3.25.*



Σχήμα 3.10 *Οι λύσεις για τις δύο εκδοχές της T defaultTerrain3. Για τη διακριτή h=5 και την ημι-συνεχή h=4.667.*



Σχήμα 3.11 *Οι λύσεις για τις δύο εκδοχές της T optimization. Για τη διακριτή h=0.699 και την ημι-συνεχή h=0.659.*

Κεφάλαιο 4. Επίλογος

Στη παραπάνω διπλωματική εργασία, μελετήσαμε τον αλγόριθμο των Pankaj K. Agarwal et al. [1] για το πρόβλημα της φύλαξης x-μονότονης πολυγωνικής γραμμής από δύο πύργους. Συγκεκριμένα, υλοποιήσαμε τους αλγορίθμους για τη διακριτή και ημι-συνεχή εκδοχή του προβλήματος με τη γλώσσα προγραμματισμού Python. Η οπτική απόδοση των αλγορίθμων έγινε με τη βοήθεια των βιβλιοθηκών matplotlib.pyplot και tkinter της Python.

Βιβλιογραφία

[1] Pankaj K. Agarwal, Sergey Bereg, Ovidiu Daescu, Haim Kaplan, Simeon Ntafos, Micha Sharir, Binhai Zhu. Guarding a Terrain by Two Watchtowers. Algorithmica 58, 352–390 (2010)

[2] Urrutia, J.: Art gallery and illumination problems. In: Sack, J.R., Urrutia, J. (eds.) Handbook of Computational Geometry, pp. 973–1026. Elsevier, Amsterdam (2000)

[3] Bespamyatnikh, S., Chen, Z., Wang, K., Zhu, B.: On the planar two-watchtower problem. In: Proc. 7th Annu. Internat. Conf. Computing and Combinatorics, pp. 121–130 (2001)

[4] Ben-Moshe, B., Carmi, P., Katz, M.J.: Computing all large sums-of-pairs in Rn and the discrete planar two-watchtower problem. Inf. Process. Lett. 89, 137–139 (2004)