ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
Τμήμα Διδακτικής της Τεχνολογίας και Ψηφιακών Συστημάτων

ΜΕΛΕΤΗ ΚΑΙ ΑΝΑΠΤΥΞΗ ΑΛΓΟΡΙΘΜΩΝ ΤΟΠΟΘΕΤΗΣΗΣ ΑΣΥΡΜΑΤΩΝ ΑΙΣΘΗΤΗΡΩΝ ΣΕ ΔΙΚΤΥΑ ΑΣΥΡΜΑΤΩΝ ΑΙΣΘΗΤΗΡΩΝ

Κοντογιάννης Γιώργος

Η εργασία υποβάλλεται για την μερική κάλυψη των απαιτήσεων
με στόχο την απόκτηση του Μεταπτυχιακού Διπλώματος Σπουδών
στην Διδακτική της Τεχνολογίας και τα Ψηφιακά Συστήματα

Ιούνιος 2007

Περίληψη

Στην εργασία αυτή, επαληθεύουμε και επεκτείνουμε το μοντέλο δυαδικού και ακέραιου προγραμματισμού για την τοποθέτηση αισθητήρων σε δίκτυα ασύρματων αισθητήρων, χρησιμοποιώντας ρεαλιστικά μοντέλα για την διάδοση του σήματος. Η αβέβαιη κάλυψη που εισάγετε για τον υπολογισμό της πιθανότητας κάλυψης ενός στόχου από έναν αισθητήρα καλύπτει ικανοποιητικά τις περιπτώσεις εξασθένησης του σήματος. Παρόλα αυτά είναι αναγκαία η χρήση μοντέλων διάδοσης τα οποία να λαμβάνουν υπόψη την εξασθένηση και την σκίαση. Έτσι αρχικά επαληθεύουμε και βελτιστοποιούμε τα μοντέλα τέλειας και αβέβαιης κάλυψης, και στην συνέχεια επεκτείνουμε το μοντέλο αβέβαιης κάλυψης για περιορισμένη ακτίνα κάλυψης, και εισάγουμε το μοντέλο διάδοσης μεγάλης κλίμακας με συσχετισμένες τιμές.

Περιεχόμενα

Κατάλογος Πινάκων

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

Κατάλογος Σχημάτων

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

Συντομογραφίες

WSN: Wireless Sensor Network

LP: Linear Programming

IP: Integer Programming

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΑ

1. Εισαγωγή

Ένας από τους ερευνητικούς τομείς που έχουν κερδίσει ιδιαίτερη προσοχή από την επιστημονική κοινότητα τα τελευταία έτη είναι αυτός των ασύρματων αισθητήρων (WSN) [1]. Λόγω της ευελιξίας τους, και της αποτελεσματικότητας τους, μπορούν να χρησιμοποιηθούν για κάθε είδους εφαρμογή συμπεριλαμβανομένου στρατιωτικών παρατηρήσεων, τη συλλογή δεδομένων, την παρακολούθηση περιβαλλοντολογικών συνθηκών καθώς και τον εντοπισμό στόχων [2].

Ένα από τα ζητήματα μεγαλύτερου ενδιαφέροντος για τα δίκτυα ασύρματων αισθητήρων είναι αυτό της κάλυψης, το οποίο επιδρά στην ποιότητα της υπηρεσίας που μπορεί να παρέχει ένα δίκτυο αισθητήρων [3]. Πολλές από τις εφαρμογές που εξυπηρετούν επιβάλλουν την ανάπτυξη των αισθητήρων, έτσι ώστε να υπάρχει η καλύτερη δυνατή κάλυψη της περιοχής εξυπηρέτησης.

Για το λόγο αυτό ένα από τα θεμελιώδη ζητήματα στην σχεδίαση δικτύων ασύρματων αισθητήρων είναι να καθοριστεί μια αποτελεσματική στρατηγική τοποθέτησης αισθητήρων, η όποια θα ελαχιστοποιεί το κόστος και θα οδηγεί σε υψηλή κάλυψη. Αρκετοί σχεδιασμοί και διατυπώσεις του προβλήματος της κάλυψης έχουν προταθεί, χρησιμοποιώντας διάφορες προσεγγίσεις ή συνδυασμούς τόσο για την περιοχή εξυπηρέτησης όσο και για τους στόχους (ελάχιστο κόστος, μέγιστη διάρκεια ζωής, ελάχιστη κάλυψη). Παρόλα αυτά καθώς ο αριθμός των αισθητήρων και το μέγεθος του δικτύου αυξάνει, τόσο μεγαλύτερη γίνεται η ανάγκη για αποτελεσματική τοποθέτηση [4].

Έτσι, για να καλύψουμε της ανάγκες αυτές, χρησιμοποιούμε μοντέλα δυαδικού και ακέραιου προγραμματισμού [5], επαληθεύοντας και βελτιστοποιώντας τον χρόνο εκτέλεσης τους. Στην συνέχεια επεκτείνουμε τη χρήση τους με ρεαλιστικά μοντέλα διάδοσης στο ασύρματο κανάλι που λαμβάνουν υπόψη την εξασθένηση και την σκίαση.

2. Βιβλιογραφική Επισκόπηση

2.1 Εισαγωγή

Πολλές από τις εφαρμογές που εξυπηρετούν τα δίκτυα αισθητήρων επιβάλλουν την σωστή τοποθέτηση τους, για την κάλυψη της περιοχή εξυπηρέτησης. Το πρόβλημα της κάλυψης αποτελεί τον κύριο παράγοντα επίδρασης στην ποιότητα της υπηρεσίας ενός δικτύου ασύρματων αισθητήρων [3]. Για το πρόβλημα αυτό έχουν προταθεί διάφοροι αλγόριθμοι τοποθέτησης, καθένας από τους οποίους διαφοροποιείται ανάλογα με την προσέγγιση του χώρου, τον μηχανισμό τοποθέτησης και τα κριτήρια κάλυψης.

2.2 Πρόβλημα Κάλυψης

Το πρόβλημα της κάλυψης απαντά στα ερωτήματα, σχετικά με την ποιότητα κάλυψης της περιοχής ενδιαφέροντος, και μπορεί να χωριστεί σε δυο κατηγορίες: αιτιοκρατικό και στοχαστικό [6]. Για κάθε μια από τις κατηγορίες αυτές, έχουν αναπτυχθεί διαφορετικοί αλγόριθμοι τοποθέτησης, βασισμένοι στα χαρακτηριστικά του κάθε μοντέλου κάλυψης.

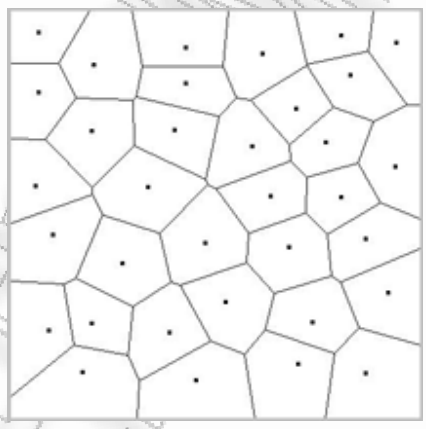
2.2.1 Αιτιοκρατική Κάλυψη

Προκειμένου να επιτευχθεί η αιτιοκρατική (deterministic) κάλυψη, θα πρέπει να αναπτυχθεί ένα στατικό δίκτυο με προκαθορισμένη μορφή. Οι προκαθορισμένες θέσεις των αισθητήρων μπορεί να είναι ομοιόμορφα κατανεμημένες στους τομείς του πεδίου ή και όχι, ανάλογα με τις ανάγκες κάλυψης. Βασικό παράδειγμα από μια ομοιόμορφη αιτιοκρατική κάλυψη είναι το πλέγμα (grid) αισθητήρων, όπου οι κόμβοι βρίσκονται στα σημεία ενός πλέγματος. Στην περίπτωση αυτή, το πρόβλημα της κάλυψης από τον αισθητήρα μοντελοποιείται με βάση τα κελιά του πλέγματος.

2.2.2 Στοχαστική Κάλυψη

Σε πολλές καταστάσεις, η αιτιοκρατική επέκταση δεν είναι εφικτή αλλά ούτε και πρακτική. Μια άλλη επιλογή ανάπτυξης του δικτύου είναι τυχαία τοποθέτηση των αισθητήρων στο περιβάλλον. Η πιθανολογική τυχαία διανομή μπορεί να ακολουθεί Gaussian, Poisson ή οποιοιδήποτε άλλη σχετική κατανομή. Παράδειγμα εφαρμογής του μοντέλου αυτού, έχουμε στην περίπτωση ρίψης των αισθητήρων για περιβαλλοντολογικές μετρήσεις.

Στην περίπτωση εφαρμογής του μοντέλου αυτού, για τον υπολογισμό της κάλυψης χρησιμοποιούνται διαγράμματα Voronoi [7]. Τα διαγράμματα Voronoi, είναι μια γραφική απεικόνιση η οποία διαχωρίζει την περιοχή σε πολύγωνα (Σχήμα 1). Κάθε αισθητήρας μπορεί να καλύπτει ένα ή περισσότερα πολύγωνα.



Σχήμα 1 - Διάγραμμα Voronoi

2.3 Κάλυψη Πλέγματος σε αιτιοκρατικά δίκτυα αισθητήρων με χρήση Θεωρίας Κωδίκων

Για την τοποθέτηση των αισθητήρων σε αιτιοκρατικά δίκτυα , μπορεί να χρησιμοποιηθεί η θεωρία κωδίκων (coding theory) [8]. Σε κάθε αισθητήρα γίνετε απόδοση μιας κωδικολέξης (codeword) η οποία και τον χαρακτηρίζει. Η τοποθέτηση των αισθητήρων διαφορικού τύπου σε ένα πλέγμα σημείων, καθορίζετε έτσι ώστε να έχουμε την επιθυμητή κάλυψη με το μικρότερο κόστος.

2.3.1 Τοποθέτηση με ελάχιστο κόστος

Έστω ότι το πεδίο των αισθητήρων αποτελείται από n_x, n_y και n_z σημεία πλέγματος στις διαστάσεις x, y, z αντίστοιχα. Υποθέτουμε ότι είναι διαθέσιμοι δυο τύποι αισθητήρων A και B με κόστος C_A και C_B, και εύρος R_A και R_B αντίστοιχα.

Ένας αισθητήρας με εύρος R_A ο οποίος τοποθετείτε σε ένα σημείο (x_1, y_1, z_1) μπορεί να εντοπίσει ένα στόχο στο σημείο (x_2, y_2, z_2), εάν η απόσπαση μεταξύ δυο σημείων $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$ είναι μικρότερη από R_A. Κάθε σημείο του πλέγματος θα πρέπει να καλύπτεται από m αισθητήρες

Θεωρούμε την δυαδική μεταβλητή a_{ijk} ως

$$a_{ijk} = \begin{cases} 1, & \text{εάν ένας αισθητήρας τύπου Α τοποθετηθεί στο σημείο } (i, j, k) \\ 0, & \text{διαφορετικά} \end{cases} \quad (1)$$

Θεωρούμε την δυαδική μεταβλητή b_{ijk} ως

$$b_{ijk} = \begin{cases} 1, & \text{εάν ένας αισθητήρας τύπου Β τοποθετηθεί στο σημείο } (i, j, k) \\ 0, & \text{διαφορετικά} \end{cases} \quad (2)$$

Το συνολικό κόστος είναι $C = \sum_{i=1}^{n_x}\sum_{j=1}^{n_y}\sum_{k=1}^{n_z}(C_A a_{ijk} + C_B b_{ijk})$ (3)

Έστω $\text{cov}^A\left((i_1, j_1, k_1),(i_2, j_2, k_2)\right)$ η κάλυψη στο σημείο (i_2, j_2, k_2) όταν ένας αισθητήρας Α τοποθετηθεί στο σημείο (i_1, j_1, k_1).

$$\text{cov}^A\left((i_1, j_1, k_1),(i_2, j_2, k_2)\right) = \begin{cases} 1, \text{ εάν ο αισθητήρας Α τοποθετηθεί στο σημείο } (i_1, j_1, k_1) \\ \quad \text{και καλύπτει το σημείο } (i_2, j_2, k_2) \\ 0, \text{ αλλιώς} \end{cases} \quad (4)$$

Ομοίως για τον αισθητήρα τύπου Β

$$\text{cov}^B\left((i_1, j_1, k_1),(i_2, j_2, k_2)\right) = \begin{cases} 1, \text{ εάν ο αισθητήρας Β τοποθετηθεί στο σημείο } (i_1, j_1, k_1) \\ \quad \text{και καλύπτει το σημείο } (i_2, j_2, k_2) \\ 0, \text{ αλλιώς} \end{cases} \quad (5)$$

Το μοντέλο ακέραιου προγραμματισμού για την κάλυψη του πλέγματος με το ελάχιστο κόστος κάλυψης είναι:

Ελαχιστοποίηση $C = \sum_{i=1}^{n_x}\sum_{j=1}^{n_y}\sum_{k=1}^{n_z}(C_A a_{ijk} + C_B b_{ijk})$

Subject to

$$\sum_{i=1}^{n_x}\sum_{j=1}^{n_y}\sum_{k=1}^{n_z} a_{i_1, j_1, k_1}\,\text{cov}^A\left((i_1, j_1, k_1),(i_2, j_2, k_2)\right) + b_{i_1, j_1, k_1}\,\text{cov}^B\left((i_1, j_1, k_1),(i_2, j_2, k_2)\right) \geq m$$

$$1 \leq i_2 \leq n_x, 1 \leq j_2 \leq n_y, 1 \leq k_2 \leq n_z$$

Οι μεταβλητές a_{ijk} και b_{ijk} είναι οι μεταβλητές απόφασης (decision variables).

Έστω $d\left(\left(i_1,j_1,k_1\right),\left(i_2,j_2,k_2\right)\right)=\sqrt{\left(i_1-i_2\right)^2+\left(j_1-j_2\right)^2+\left(k_1-k_2\right)^2}$ η απόσταση μεταξύ των σημείων (i_1,j_1,k_1) και (i_2,j_2,k_2). Εφόσον $\text{cov}^A\left(\left(i_1,j_1,k_1\right),\left(i_2,j_2,k_2\right)\right)=1$ εάν ο αισθητήρας τύπου Α τοποθετηθεί στο σημείο (i_1,j_1,k_1) και καλύπτει το σημείο (i_2,j_2,k_2) εισάγουμε τις ακόλουθες ανισότητες

$$\text{cov}^A\cdot(R_A-d)\geq 0 \qquad (6)$$

$$(1-\text{cov}^A)\cdot(d-R_A)\geq 0 \qquad (7)$$

Εάν $d>R_A$, τότε η κάλυψη cov^A πρέπει να είναι μηδέν για να ισχύει η ανισότητα (6), ενώ αν $d<R_A$, τότε η κάλυψη cov^A πρέπει να είναι 1, για να ικανοποιείται η ανισότητα (7)

Οι περιορισμοί δεν είναι γραμμικοί εφόσον περιέχουν αποτελέσματα δυαδικών μεταβλητών. Έστω u και υ δυαδικές μεταβλητές. Το αποτέλεσμα $u\upsilon$ μπορεί να αντικατασταθεί από μια καινούργια μεταβλητή ω αρκεί να ικανοποιούνται οι παρακάτω συνθήκες

$$u+\upsilon\geq 2\omega \qquad (8)$$

$$u+\upsilon-1\leq\omega \qquad (9)$$

Έτσι τα γινόμενα στον περιορισμό της αντικειμενικής συνάρτησης για την ελαχιστοποίηση του κόστους μπορούν να γραφτούν ως

$$a_{i_1,j_1,k_1}\,\text{cov}^A\left(\left(i_1,j_1,k_1\right),\left(i_2,j_2,k_2\right)\right) => g((i_1,j_1,k_1),(i_2,j_2,k_2)) \quad (9)$$

$$b_{i_1,j_1,k_1}\,\text{cov}^B\left(\left(i_1,j_1,k_1\right),\left(i_2,j_2,k_2\right)\right) => h((i_1,j_1,k_1),(i_2,j_2,k_2)) \quad (10)$$

Ικανοποιώντας τις συνθήκες (3), (4). Έτσι το μοντέλο γραμμικού και ακέραιου προγραμματισμού μπορεί να γραφεί ως

Ελαχιστοποίηση $C = \sum_{i=1}^{n_x}\sum_{j=1}^{n_y}\sum_{k=1}^{n_z}(C_A a_{ijk} + C_B b_{ijk})$:

Subject to

11. $\text{cov}\, A \cdot (R_A - d) \geq 0$

    για όλα τα ζευγάρια σημείων του πλέγματος με βάση την ανισότητα (6)

12. $(1 - \text{cov}^A) \cdot (d - R_A) \geq 0$

    για όλα τα ζευγάρια σημείων του πλέγματος με βάση την ανισότητα (7)

13. $\text{cov}^B \cdot (R_B - d) \geq 0$

    για όλα τα ζευγάρια σημείων του πλέγματος με βάση την ανισότητα (6)

14. $(1 - \text{cov}^B) \cdot (d - R_B) \geq 0$

    για όλα τα ζευγάρια σημείων του πλέγματος με βάση την ανισότητα (7)

15. $$\sum_{i_1=1}^{n_x}\sum_{j_1=1}^{n_y}\sum_{k_1=1}^{n_z}\left(g((i_1,j_1,k_1),(i_2,j_2,k_2)) + h((i_1,j_1,k_1),(i_2,j_2,k_2))\right) \geq m$$
$$1 \leq i_2 \leq n_x, 1 \leq j_2 \leq n_y, 1 \leq k_2 \leq n_z$$

    Με βάση τις (8), (9), και την ικανοποίηση των συνθηκών που ακολουθούν

16. $$\text{cov}^A((i_1,j_1,k_1),(i_2,j_2,k_2)) + d((i_1,j_1,k_1),(i_2,j_2,k_2)) \geq 2g((i_1,j_1,k_1),(i_2,j_2,k_2)),$$
$$1 \leq i_2 \leq n_x, 1 \leq j_2 \leq n_y, 1 \leq k_2 \leq n_z$$

    Με βάση την ανισότητα (8) για την περίπτωση του αισθητήρα τύπου Α

17. $$\text{cov}^A((i_1,j_1,k_1),(i_2,j_2,k_2)) + d((i_1,j_1,k_1),(i_2,j_2,k_2)) - 1 \leq g((i_1,j_1,k_1),(i_2,j_2,k_2)),$$
$$1 \leq i_2 \leq n_x, 1 \leq j_2 \leq n_y, 1 \leq k_2 \leq n_z$$

    Με βάση την ανισότητα (8) για την περίπτωση του αισθητήρα τύπου Α

18. $$\text{cov}^B((i_1,j_1,k_1),(i_2,j_2,k_2)) + d((i_1,j_1,k_1),(i_2,j_2,k_2)) \geq 2g((i_1,j_1,k_1),(i_2,j_2,k_2)),$$
$$1 \leq i_2 \leq n_x, 1 \leq j_2 \leq n_y, 1 \leq k_2 \leq n_z$$

    Με βάση την ανισότητα (9) για την περίπτωση του αισθητήρα τύπου Β

19. $$\text{cov}^B((i_1,j_1,k_1),(i_2,j_2,k_2)) + d((i_1,j_1,k_1),(i_2,j_2,k_2)) - 1 \leq g((i_1,j_1,k_1),(i_2,j_2,k_2)),$$
$$1 \leq i_2 \leq n_x, 1 \leq j_2 \leq n_y, 1 \leq k_2 \leq n_z$$

    Με βάση την ανισότητα (9) για την περίπτωση του αισθητήρα τύπου Β

20. $a_{ijk}, b_{ijk} = 0$ ή $1 \leq i_2 \leq n_x, 1 \leq j_2 \leq n_y, 1 \leq k_2 \leq n_z$

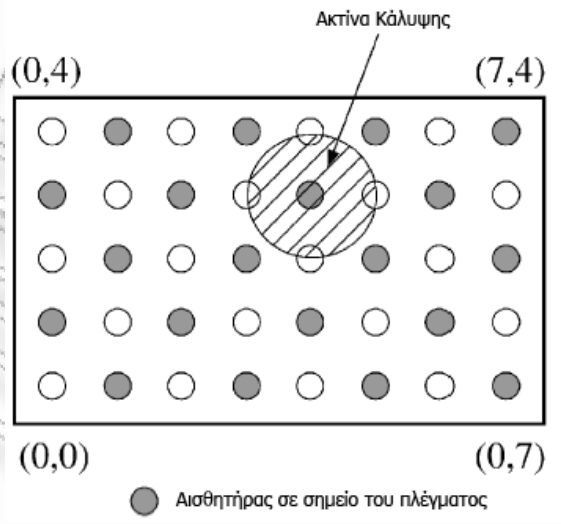
21. $a_{ijk} + b_{ijk} \leq 1, 1 \leq i_2 \leq n_x, 1 \leq j_2 \leq n_y, 1 \leq k_2 \leq n_z$

2.3.2 Τοποθέτηση Αισθητήρων

Το πρόβλημα τοποθέτησης αισθητήρων στα σημεία του πλέγματος, αντιμετωπίζεται έτσι ώστε κάθε σημείο να μπορεί να οριστεί μοναδικά από ένα υποσύνολο αισθητήρων που μπορούν να εντοπίσουν το στόχο. Αυτή η προσέγγιση βασίζεται στην ιδέα της απόδοσης μοναδικών κωδικών για μοναδικό προσδιορισμό των κορυφών

Η απόδοση των κωδικών μπορεί να οριστεί ως η ιδανική κάλυψη των κορυφών ενός γράφου G έτσι ώστε κάθε κορυφή του να μπορεί να οριστέι μοναδικά από τις κορυφές που την καλύπτουν. Η σφαίρα της ακτίνα κάλυψης r με κέντρο, το κέντρο της κορυφής u ορίζεται ως το σύνολο των κορυφών σε απόσταση μικρότερη ή ίση από r, με την απόσταση να ορίζεται ως ο αριθμός των ακμών στο μικρότερο μονοπάτι. Κάθε κορυφή θεωρούμε ότι καλύπτει τον εαυτό της.

Το πρόβλημα κάλυψης με βάση την χρήση την θεωρία κωδίκων, ορίζεται ως εξής: για ένα γράφο G, θα πρέπει να βρεθεί ο ελάχιστος αριθμός κορυφών C έτσι ώστε κάθε κορυφή G να ανήκει σε ένα μοναδικό σύνολο σφαιρών με κέντρο τις κορυφές του C. Στο σύνολο των κορυφών αποδίδουμε και ένα κωδικό για αναγνώριση.



Σχήμα 2 – Σημεία Πλέγματος

2.4 Αλγόριθμος Ελαχίστου-Μεγίστου

Η τοποθέτηση αισθητήρων σε δισδιάστατα και τρισδιάστατα [20] πλέγματα στον αλγόριθμο με απόδοση κωδίκων, έχει διατυπωθεί ως συνδυαστικό πρόβλημα βελτιστοποίησης, με χρήση ακέραιου γραμμικού προγραμματισμού. Η προσέγγιση αυτή πάσχει από δύο κύρια μειονεκτήματα. Κατ' αρχάς, η υπολογιστική πολυπλοκότητα καθιστά την προσέγγιση ανέφικτη για μεγάλα δίκτυα αισθητήρων. Δεύτερον, η προσέγγιση κάλυψης πλέγματος στηρίζεται στη «τέλεια» ανίχνευση αισθητήρων.

Δεδομένης της αβέβαιης κάλυψης στο πεδίο των αισθητήρων [9], θα πρέπει να ληφθεί υπόψη η αβεβαιότητα που συνδέεται με την ανίχνευση του αισθητήρα. Την ανάγκη αυτή έρχεται να καλύψει ο αλγόριθμος ελαχίστου-μεγίστου [10].

Σκοπός του min - max αλγόριθμου είναι η ανάπτυξη ενός πλαισίου βελτιστοποίησης των περιορισμένων πόρων των αισθητήρων, κάτω από τον περιορισμό της κάλυψης του πεδίου. Η προτεινόμενη προσέγγιση, αποσκοπεί στην χρήση του ελάχιστου αριθμού αισθητήρων σε συνδυασμό με την μικρότερη δυνατή αποστολή δεδομένων.

2.4.1 Μοντέλο Εδάφους

Θεωρούμε ότι το πεδίο των αισθητήρων αποτελείται από σημεία πλέγματος, η πυκνότητα των οποίων καθορίζεται από την ακρίβεια τοποθέτησης. Η πιθανότητα κάλυψης είναι εκθετική με εκθέτη την απόσταση μεταξύ αισθητήρα και στόχου. Έτσι ένα σημείο σε απόσταση d από τον αισθητήρα εντοπίζεται με πιθανότητα e^{-ad}. Η παράμετρος a μπορεί να χρησιμοποιηθεί για να μοντελοποιήσει την ποιότητα του αισθητήρα και το εύρος στο οποίο η πιθανότητα εντοπισμού μειώνεται με την απόσταση. Η πιθανότητα εντοπισμού είναι 1 εάν η θέση του στόχου και ο αισθητήρας βρίσκονται στο ίδιο σημείο. Για κάθε δυο σημεία i και j στο πεδίο του αισθητήρα, ορίζουμε δυο πιθανότητες:

1. p_{ij} πιθανότητα ο στόχος στο σημείο j εντοπίζεται από ένα αισθητήρα στο σημείο i
2. p_{ji} η πιθανότητα ο στόχος στο σημείο i να εντοπίζεται από τον αισθητήρα στο σημείο j.

Οι πιθανότητες αυτές είναι συμμετρικές και ισχύει ότι $p_{ij} = p_{ji}$. Στην περίπτωση που υπάρχει κάποιο εμπόδιο μεταξύ των δυο σημείων $p_{ij} = 0$

Ο σκοπός του αλγορίθμου τοποθέτησης είναι να προσδιορίσει τον ελάχιστο αριθμό αισθητήρων και την θέση τους έτσι ώστε κάθε σημείο να καλύπτεται με το ελάχιστο επίπεδο εμπιστοσύνης. Το κατώφλι κάλυψης T παρέχεται σαν είσοδος στον αλγόριθμο τοποθέτησης . Στόχος είναι να εξασφαλιστεί ότι κάθε σημείο του πλέγματος καλύπτεται με πιθανότητα τουλάχιστον T.

Η συνάρτηση MAX_AVG_COV προσπαθεί να μεγιστοποιήσει την μέση κάλυψη των σημείων του πλέγματος ενώ η συνάρτηση MAX_MIN_COV προσπαθεί να μεγιστοποιήσει την κάλυψη των σημείων που δεν καλύπτονται επαρκώς.

Ξεκινάμε την διαδικασία σχηματίζοντας τον πίνακα εντοπισμού $D = [p_{ij}]$ για κάθε ζευγάρι των σημείων του πλέγματος στο πεδίο του αισθητήρα. Για κάθε $n \times n$ πλέγμα έχουμε συνολικά n^2 γραμμές και στήλες, συνολικά n^4 στοιχεία.

Από τον πίνακα εντοπισμού D, καθορίζουμε την πιθανότητα απώλειας $M = m_{ij}$ όπου $m_{ij} = 1 - p_{ij}$. Και οι δυο συναρτήσεις χρησιμοποιούν την ευρεστική μέθοδο Greedy [17] για τον καθορισμό της καλύτερης θέσης την συγκεκριμένη χρονική στιγμή. Οι αλγόριθμοι είναι επαναληπτικοί και σε κάθε επανάληψη τοποθετούν έναν αισθητήρα. Ο αλγόριθμος τερματίζει όταν η κάλυψη των σημείων έχει ολοκληρωθεί ή ένας ικανοποιητικός αριθμός αισθητήρων έχει τοποθετηθεί.

Ορίζουμε ένα διάνυσμα $M^{*} = (M_1, M_2, .., M_N)$ με τις πιθανότητες απωλειών των $N = n^2$ σημείων του πλέγματος. Μια πιθανότητα απώλειας M_i στο διάνυσμα αυτό δείχνει πως το σημείο i δεν καλύπτεται επαρκώς. Το διάνυσμα αυτό αρχικοποιείται $M^{*} = (1,1,...,1)$. Κάθε φορά που ένας αισθητήρας τοποθετείται πεδίο των αισθητήρων, μειώνει μια ή περισσότερες τιμές του ανύσματος.

Για να μοντελοποιήσουμε την επιλεκτική τοποθέτηση των αισθητήρων εισάγουμε την πιθανότητα προστασίας pr_i σε κάθε σημείο i. Το κατώφλι της πιθανότητας απώλειας για το σημείο πλέγματος i εκφράζετε ως $M^{i}_{\min} = 1 - pr_i$.

Μια εναλλακτική μέθοδος που μπορεί να χρησιμοποιηθεί είναι η τοποθέτηση σε κάθε επανάληψη ενός αισθητήρα στο σημείο με την μικρότερη κάλυψη. Με τον τρόπο αυτό σε κάθε επανάληψη καλύπτεται το σημείο με την μικρότερη κάλυψη. Η διαδικασία αυτή πραγματοποιείται έως ότου η κάλυψη των σημείων έχει ολοκληρωθεί ή το κατώφλι κάλυψης έχει ξεπεραστεί.

Έστω ότι η απόσταση μεταξύ δυο σημείων P_1 και P_2 είναι d και η απόσταση μεταξύ γειτονικών αισθητήρων είναι d^{*}. Εάν η τιμή $\left(d + d*/\sqrt{2}\right)$ χρησιμοποιηθεί για να υπολογίσει την κάλυψη του σημείου P_1 ως προς το σημείο P_2 και ο αριθμός των διαθέσιμων αισθητήρων είναι επαρκής, η πιθανότητα απώλειας όλων των σημείων εκτός πλέγματος είναι μικρότερη από το κατώφλι $M_{\min}$ όταν οι αλγόριθμοι MAX_AVG_COV και MIN_AVG_COV έχουν ολοκληρωθεί.

2.4.2 Μεγιστοποίηση μέσης κάλυψης

Procedure MAX_ AVG _COV (M, M^*,$\mathrm{M}_{\min}$)
begin
num_sensors:=1;
repeat
for $i := 1$ to N do
$\sum_i m_{i1} + m_{i2} + \ldots + m_{iN;}$
Place sensor at grid point k such that $\sum_k$ is minimum;
for $i := 1$ to N do
$M_i = M_i m_{ki}$;
Delete k^{th} row and column from the M matrix
num_sensors:=num_sensors+1;
until $M_i < M_{\min}$ for all i, $1 \le i \le N$ or num_sensors>limit
end

2.4.3 Μεγιστοποίηση σημείων με μη επαρκή κάλυψη

Procedure MAX_MIN_COV (M, M^*,$\mathrm{M}_{\min}$)
begin
Place first sensor randomly
num_sensors:=1;
repeat
for $i := 1$ to N do
$M_i = M_i m_{ki}$
Place sensor at grid point k such that M_k is max
Delete k^{th} row and column from the M matrix
num_sensors:=num_sensors+1;
until $M_i < M_{\min}$ for all i, $1 \le i \le N$ or num_sensors>limit
end

2.5 Αλγόριθμος Εικονικής Δύναμης

Προκειμένου να καλυφθεί τόσο η περίπτωση της τέλειας όσο και τις πιθανοτικής κάλυψης που εισήγαγαν οι δυο προηγούμενοι αλγόριθμοι, αλλά και να μειωθεί σε μεγαλύτερο βαθμό η ανάγκη για υπολογιστική ισχύ, αναπτύχθηκε ο αλγόριθμος της Εικονικής Δύναμης (Virtual Force) [11]. Η κύρια ιδέα στην ανάπτυξη του αλγορίθμου είναι ότι η τυχαία κάλυψη από αισθητήρες μπορεί να βελτιωθεί με την χρήση αλγορίθμου κατευθυνόμενου από δύναμη. Η χρήση του αλγορίθμου λαμβάνει χώρα μετά από την τυχαία τοποθέτηση των αισθητήρων σε δισδιάστατο πεδίο.

Ο εντοπισμός του στόχου στηρίζεται στην επικοινωνία μεταξύ των αισθητήρων και του επικεφαλής της συστάδας (cluster) η οποία πραγματοποιείται σε δυο στάδια. Στο πρώτο στάδιο οι αισθητήρες ενημερώνουν τον επικεφαλής της συστάδας για την θέση του στόχου μεταφέροντας την ελάχιστη απαιτούμενη πληροφορία. Στο δεύτερο στάδιο ο επικεφαλής αναλύει την πληροφορία αυτή προσδιορίζοντας την θέση του στόχου.

Για το δίκτυο αισθητήρων σε συστάδες θεωρούμε τα ακόλουθα:

- Μετά την τυχαία τοποθέτηση, όλοι οι αισθητήρες είναι σε θέση να επικοινωνήσουν με τον επικεφαλής της συστάδας
- Ο επικεφαλής της συστάδας είναι αυτός που αποφασίζει την εφαρμογή του αλγορίθμου και την κίνηση ενός αισθητήρα.
- Οι αισθητήρες ενημερώνουν τον επικεφαλής της συστάδας με πληροφορία ναι / όχι. Ο επικεφαλής της συστάδας συλλέγει περισσότερες πληροφορίες με ερωτήματα ανά τομέα αισθητήρων.

Κάθε αισθητήρας συμπεριφέρεται σαν πηγή δύναμης για κάθε άλλο αισθητήρα. Η δύναμη αυτή μπορεί να είναι είτε θετική είτε αρνητική ανάλογα με την απόσταση τους. Το κριτήριο αυτό ορίζεται από ένα προκαθορισμένο κατώφλι, το οποίο όταν ξεπερνάται, η δύναμη είναι αρνητική. Αντίστοιχα για μεγάλες αποστάσεις η δύναμη είναι θετική.

Σε ένα πεδίο n επί m αισθητήρων υποθέτουμε ότι υπάρχουν k αισθητήρες. Κάθε αισθητήρας έχει εύρος κάλυψης d. Υποθέτουμε ότι τοποθετούμε τον αισθητήρα s_i στην θέση (x_i, y_i). Για κάθε σημείο P στην θέση (x, y) η ευκλείδεια απόσταση μεταξύ s_i και P είναι $d(s_i, P) = \sqrt{(x_i - x)^2 - (y_i - y)^2}$.

Η κάλυψη του αισθητήρα είναι

$$c_{xy}(s_i) = \begin{cases} 1, & \text{εάν } d(s_i, P) < r \\ 0, & \text{διαφορετικά} \end{cases} \quad (23)$$

Εάν θεωρήσουμε ότι η κάλυψη είναι πιθανοτική έχουμε

$$c_{xy}(s_i) = \begin{cases} 0, & r + r_e < d(s_i, P) \\ e^{-\lambda a^{\beta}}, & r - r_e < d(s_i, P) < r + r_e \\ 1, & r - r_e \geq d(s_i, P) \end{cases} \quad (24)$$

Όπου $r_e (r_e < r)$ είναι το μέτρο αβεβαιότητας στον εντοπισμό του αισθητήρα, $a = d(s_i, P) - (r - r_e)$ και α, β οι παράμετροι υπολογισμού της πιθανότητας εντοπισμού όταν ο στόχος είναι σε απόσταση μεγαλύτερη από r_e, αλλά μέσα στα όρια κάλυψης του αισθητήρα.

2.5.1 Εικονική Δύναμη

$$\vec{F}_i = \sum_{j=1, j\neq i}^{k} \vec{F}_{ij} + \vec{F}_{iR} + \vec{F}_{iA} \quad (25)$$

$\vec{F}_i$ η δύναμη που ασκείται στον αισθητήρα s_i

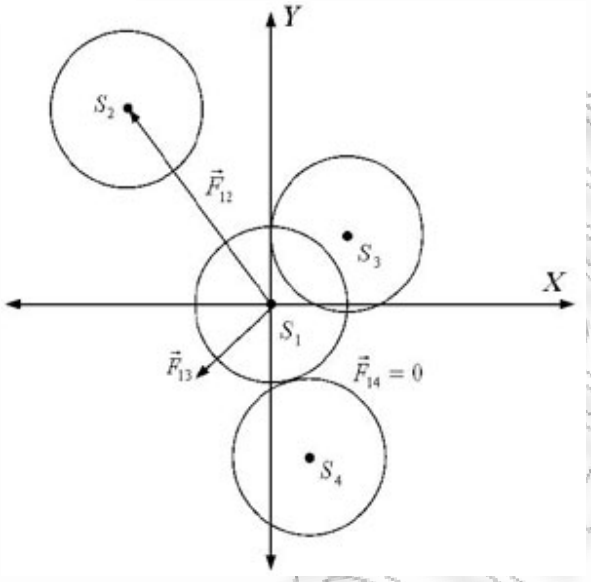
$\vec{F}_{iA}$ η συνολική δύναμη που ασκείται στον αισθητήρα s_i από γειτονικές περιοχές κάλυψης

$\vec{F}_{iR}$ η συνολική δύναμη που ασκείται στον αισθητήρα s_i από εμπόδια.

$$\vec{F}_{ij} = \begin{cases} (\omega_A(d_{ij} - d_{ih})), & if\ d_{ij} > d_{th} \\ 0, & if\ d_{ij} = d_{th} \\ \omega_R \frac{1}{d_{ij}}, a_{ij} + \pi & if\ otherwise \end{cases} \quad (26)$$

$\vec{F}_{ij}$ είναι η δύναμη που ασκείται στον αισθητήρα s_i από έναν άλλο s_j.

Όπου d_{ij} είναι η ευκλείδεια απόσταση μεταξύ των αισθητήρων s_i και s_j, d_{th} είναι το κατώφλι της απόστασης της απόστασης μεταξύ των αισθητήρων s_i και s_j, a_{ij} είναι ο προσανατολισμός (γωνία) της ευθείας γραμμής μεταξύ των αισθητήρων s_i και s_j και ω_R είναι το μέτρο της δύναμης έλξης.



Σχήμα 3 – Παράδειγμα 4 αισθητήρων

Εάν $re \approx 0$ χρησιμοποιούμε δυαδικό μοντέλο, προσπαθώντας να κάνουμε την απόσταση d_{ij} όσο το δυνατόν πιο κοντά σε $2r$. Αυτό εξασφαλίζει ότι δεν θα υπάρχουν επικαλυπτόμενες περιοχές., κερδίζοντας έτσι σε αριθμό αισθητήρων αλλά και ισχύος. Μειονέκτημα του μοντέλου αυτού είναι ότι θα υπάρχουν σημεία τα οποία δεν θα καλύπτονται. Για να αποφύγουμε το πρόβλημα αυτό μπορούμε να επιτρέψουμε επικαλυπτόμενες περιοχές, εξασφαλίζοντας την πλήρη κάλυψη αλλά με μεγαλύτερο αριθμό αισθητήρων.

Για $r_e > 0$, το r_e δεν είναι πλέον αμελητέο. Στην περίπτωση αυτή τα σημεία του πλέγματος δεν καλύπτονται όλα με την ίδια πιθανότητα, έτσι κρίνεται αναγκαία η ύπαρξη επικαλυπτόμενων περιοχών. Θεωρούμε ένα σημείο (x, y) σε μια επικαλυπτόμενη περιοχή από τους αισθητήρες s_i και s_j. Η πιθανότητα να αναφερθεί ένα στόχος στην περιοχή αυτή είναι $c_{xy}(s_i, s_j)$. Οι αισθητήρες ανήκουν σε στιβάδα και ενεργούν ανεξάρτητα. Έτσι

$$c_{xy}(s_i, s_j) = 1 - (1 - c_{xy}(s_i))(1 - c_{xy}(s_j)) \quad (27)$$

Αν c_{th} είναι το επιθυμητό κατώφλι για όλα τα σημεία του πλέγματος, τότε θα πρέπει

$$\min_{x,y}\left\{c_{x,y}(s_i,s_y)\right\} \geq c_{th} \quad (28)$$

2.5.2 Πίνακας Πιθανοτήτων Εντοπισμού

Μετά την ολοκλήρωση του Virtual Force αλγορίθμου, οι αισθητήρες έχουν τοποθετηθεί στην τελική τους θέση, και ο επικεφαλής της συστάδας δημιουργεί τον πίνακα πιθανοτήτων εντοπισμού, για κάθε σημείο του πλέγματος. Ο πίνακας αυτός περιέχει όλες τις πιθανές αναφορές εντοπισμού από τους αισθητήρες εκείνους που μπορούν να εντοπίσουν ένα στόχο.

Θεωρούμε ότι ένα σημείο $P(x,y)$ καλύπτεται από ένα σύνολο k_{xy} αισθητήρων με $S_{xy}=k_{xy}$ για $0\leq k_{xy}\leq k$ και $S_{xy}\subseteq\{s_1,s_2,..,s_k\}$. Υπάρχουν $2^{k_{xy}}$ πιθανότητες για k_{xy} αισθητήρες να αναφερθεί ένα γεγονός. Στις πιθανότητες αυτές συμπεριλαμβάνεται και η πιθανότητα κανένας από τους αισθητήρες να μην εντοπίσει στόχο (“00..0”) ή και όλοι (“11…1”). Έτσι ο πίνακας πιθανοτήτων ορίζετε ως

$$p\_table(i)=\prod_{s_j\in S_{x,y}} p_{xy}(s_j,i) \quad (29)$$

Όπου $0\leq i\leq 2^{k_{xy}}$ και $p_{xy}(s_j,i)=c_{x,y}(s_j)$ εάν ένας αισθητήρας s_j εντοπίσει ένα στόχο στο σημείο $P(x,y)$ διαφορετικά $p_{xy}(s_j,i)=1-c_{x,y}(s_j)$

2.5.3 Βαθμονόμηση

Μετά την δημιουργία του πίνακα πιθανοτήτων, ο προσδιορισμός γίνεται από τον επικεφαλής της συστάδας εάν ο στόχος έχει εντοπιστεί από παραπάνω από ένα

αισθητήρες. Ο επικεφαλής της συστάδας γνωρίζει με βάση τον πίνακα πιθανοτήτων ποιος αισθητήρας είναι ο κατάλληλος για να δώσει περισσότερες πληροφορίες για τον στόχο.

Θεωρούμε $S_{rep}(t)$ το σύνολο των αισθητήρων που έχουν αναφέρει τον εντοπισμό ενός αντικειμένου $S_{rep,xy}(t)$ στην θέση $P(x,y)$.

$$S_{rep,xy}(t) \subseteq S_{rep}(t) \text{ και } S_{rep,xy}(t) \subseteq S_{xy} \quad (30)$$

Το βάρος του σημείου $P(x,y)$ την χρονική στιγμή t είναι $w_{xy}(t) = \frac{k_{rep,xy}(t)}{k_{rep}(t)}$ (31)

Όπου $k_{rep}(t) = \left|S_{rep}(t)\right|$ και $k_{rep,xy}(t) = \left|S_{rep,xy}(t)\right|$

Το score στο σημείου $P(x,y)$ την χρονική στιγμή t είναι

$SCORE_{xy}(t) = p\_table_{xy}(i(t)) \times w_{xy}(t)$ (32)

2.5.4 Επιλογή σημείων για ερώτηση

Θεωρούμε ότι ο μέγιστος αριθμός αισθητήρων που επιτρέπεται να αναφέρει ένα γεγονός είναι $k_{\max}$ και το σύνολο των αισθητήρων που θα ρωτήσει ο επικεφαλής της συστάδας την χρονική στιγμή t είναι $S_q(t)$. Σύμφωνα με τα αποτελέσματα των αναφορών του αισθητήρα και της διαδικασίας εντοπισμού, για μια χρονική στιγμή t, εάν $k_{\max} \geq k_{rep}(t)$ τότε όλοι οι αισθητήρες μπορούν να ερωτηθούν. Διαφορετικά επιλέγουμε αισθητήρες με βάση την αξιολόγηση τους.

2.6 Τοποθέτηση αισθητήρων βοηθούμενων από κίνηση

Η προηγούμενη μέθοδος τοποθέτησης αισθητήρων στηρίζεται σε κεντρικές προσεγγίσεις. Σε πολλές περιπτώσεις τοποθέτησης όμως, η ύπαρξη κεντρικού συστήματος δεν είναι διαθέσιμη. Είναι επίσης πιθανό να μην είναι δυνατός ο έλεγχος σε συστάδες λόγο της αδυναμίας κατανομής του δικτύου.

Τα δυο πρωτόκολλα που προτείνονται για τον έλεγχο της κίνησης των αισθητήρων με σκοπό την κάλυψη του στόχου, είναι το «βασικό πρωτόκολλο» και το «πρωτόκολλο εικονικής κίνησης» [12]. Στο βασικό πρωτόκολλο, οι αισθητήρες κινούνται επαναληπτικά μέχρι την τελική θέση τους. Σε κάθε επανάληψη, οι αισθητήρες εντοπίζουν τα κενά κάλυψης χρησιμοποιώντας διαγράμματα Voronoi. Εάν υπάρχει κενό, υπολογίζεται η θέση του στόχου για να καλυφθεί το κενό και να μετατοπιστεί ο αισθητήρας. Στο πρωτόκολλο εικονικής κίνησης, οι αισθητήρες δεν κινούνται φυσικά αλλά εικονικά. Μετά τον υπολογισμό της θέση των στόχων, οι αισθητήρες κινούνται εικονικά ενημερώνοντας για την θέση τους, τους υπόλοιπους γειτονικούς αισθητήρες. Η πραγματική κίνηση των αισθητήρων πραγματοποιείται εφόσον καθοριστεί η τελική τους θέση ή το κόστος επικοινωνίας των γειτονικών αισθητήρων αυξηθεί αρκετά.

Για τα δυο αυτά πρωτόκολλα αναπτύχθηκαν τρεις αλγόριθμοι υπολογισμού της θέσης του στόχου όταν υπάρχουν κενά κάλυψης. Οι αλγόριθμοι αυτοί είναι οι «Διανυσματικός Αλγόριθμος », «Αλγόριθμος Voronoi» και «Αλγόριθμός Μεγίστου – Ελαχίστου». Στον διανυσματικό αλγόριθμο οι αισθητήρες απομακρύνονται από τα σημεία με πυκνή ύπαρξη αισθητήρων. Στον Voronoi αλγόριθμο οι αισθητήρες μετακινούνται προς τα σημεία με κενά κάλυψης και τέλος στον αλγόριθμο μεγίστου – ελαχίστου οι αισθητήρες μετακινούνται ομοίως προς τα σημεία με κενά κάλυψης ελέγχοντας όμως τη δημιουργία νέων κενών κάλυψης κατά την μετατόπισης τους.

2.6.1 Διαγράμματα Voronoi

Τα διάγραμμα Voronoi είναι μια σημαντική δομή δεδομένων στην υπολογιστική γεωμετρία. Παρουσιάζει την γειτονική πληροφορία ενός συνόλου γεωμετρικών κόμβων. Σε ένα Voronoi διάγραμμα ο χώρος τεμαχίζεται σε σύνολο από κόμβους, με μορφή πολυγώνου. Ορίζεται ως Voronoi πολύγωνο του σημείου s_o το $G_0 = \langle Vo, \mathrm{E}_0 \rangle$, όπου V_0 είναι το σύνολο των κορυφών, και E_0 είναι το σύνολο των ακμών.

2.6.2 Διανυσματικός Αλγόριθμος

Ο αλγόριθμος αυτός δραστηριοποιείται βασισμένος στις ιδιότητες των ηλεκτρομαγνητικών πεδίων. Έστω $d(s_i, s_j)$ η απόσταση μεταξύ των αισθητήρων s_i και s_j. d_{ave} είναι η μέση απόσταση μεταξύ δυο αισθητήρων όταν οι αισθητήρες αυτοί είναι τοποθετημένοι στην περιοχή κάλυψης του στόχου. Η απόσταση αυτή μπορεί να υπολογιστεί από την αρχή, θωρώντας δεδομένη την θέση των αισθητήρων και του στόχου. Η εικονική δύναμη F_{ij} μεταξύ των δυο αισθητήρων θα τους απωθήσει σε απόσταση $(d_{ave} - d(s_i, s_j))/2$ μεταξύ τους. Σε περίπτωση που ο ένας αισθητήρας καλύπτει το διάγραμμα Voronoi του και δεν θα πρέπει να μετατοπιστεί, ο άλλος αισθητήρας μετακινείται κατά $d_{ave} - d(s_i, s_j)$.

Επιπρόσθετα με την εικονική δύναμη F_{ij} που αναπτύσσεται λόγο των αισθητήρων, αναπτύσσεται και η δύναμη πεδίου F_b, η οποία κινεί τους αισθητήρες πολύ κοντά στο εσωτερικό όριο. Η δύναμη F_b κινεί τον αισθητήρα κατά $d_{ave}/2 - d_b(s_i)$, όπου $d_b(s_i)$ είναι η απόσταση του s_j από το όριο. Η συνολική δύναμη που ασκείται στον αισθητήρα είναι το άθροισμα των δυνάμεων που ασκείται από το πεδίο και από τους γειτονικούς αισθητήρες.

Για να μειώσουμε τα λάθη από την εφαρμογή της εικονικής δύναμης, όταν ένας αισθητήρας καθορίσει την τελική του θέση, ελέγχει εάν η θέση αυτή αυξάνει την συνολική κάλυψη του πεδίου. Η συνολική κάλυψη ορίζεται, ως η κάλυψη του τοπικού πολυγώνου Voronoi. Εάν η τοπική κάλυψη δεν αυξηθεί, ο αισθητήρας δεν θα πρέπει να μετατοπιστεί στην θέση του αισθητήρα. Παρόλο που η κατεύθυνση της μετατόπισης είναι σωστή, μπορεί η θέση του στόχου να είναι πολύ μακριά. Στην περίπτωση αυτή επιλέγουμε το ενδιάμεσο σημείο ή το σημείο στα $3/4$ μεταξύ του στόχου και της τρέχουσας θέσης του αισθητήρα.

2.6.3 Αλγόριθμός Voronoi

Ο αλγόριθμος Voronoi είναι ένας Greedy [17] αλγόριθμος, ο οποίος καλύπτει κάθε φορά το μεγαλύτερο κενό κάλυψης. Σε αντίθεση με τον διανυσματικό αλγόριθμο, ο βασισμένος στα διαγράμματα Voronoi, είναι ελκτικός. Εφόσον ο αισθητήρας εντοπίσει την ύπαρξη κενού κάλυψης, θα μετατοπιστεί στη μακρυνότερη κορυφή του πολυγώνου, και θα σταματήσει όταν αυτή καλυφθεί. Η μέγιστη απόσταση περιορίζεται στο μισό της ακτίνας κάλυψης του αισθητήρα.

2.6.4 Αλγόριθμος Ελαχίστου – Μεγίστου

Ομοίως με τον Voronoi, καλύπτει τις περιοχές με την μεγαλύτερη κάλυψη μετακινώντας τον αισθητήρα προς την κορυφή με την μεγαλύτερη απόσταση. Η διαφορά στον αλγόριθμο αυτό είναι ότι τώρα αποφεύγεται η περίπτωση δημιουργίας νέου μεγίστου μετά την τοποθέτηση. Αυτό επιτυγχάνετε με την επιλογή σημείου μέσα στο πολύγωνο η απόσταση του οποίου O_m από το μέγιστο σημείο ελαχιστοποιείται.

2.7 Μοντέλα δυαδικού και ακέραιου προγραμματισμού και ευριτικοί αλγόριθμοι

Ο συνδυασμός υπολογιστικής γεωμετρίας και διαγραμμάτων Voronoi, παρόλο που υπολογίζει την καλυπτόμενη περιοχή προσφέροντας γραφική απεικόνιση της, μειονεκτεί μιας και θεωρεί την κάλυψη ενός αισθητήρα βέβαιη. Το γεγονός αυτό περιορίζει αρκετά το μοντέλο μιας και η ανίχνευση των αισθητήρων είναι αβέβαιη [9].

Η προσέγγιση αυτή καλύπτει και την περίπτωση ύπαρξης εμποδίου. Στην περίπτωση αυτή ο αισθητήρας δεν μπορεί να καλύψει την περιοχή και να εντοπίσει τον στόχο, ακόμα και αν είναι αρκετά κοντά. Αν και η περίπτωση της αβέβαιης κάλυψης έχει ληφθεί υπόψη στον αλγόριθμο μεγίστου, οι αισθητήρες που τοποθετούνται είναι ίδιοι. Επίσης η επίλυση του προβλήματος γίνεται ιδιαίτερα δύσκολη στην περίπτωση μεγάλων δικτύων.

Σκοπός του αλγορίθμού είναι ο καθορισμός και η βελτιστοποίηση της τοποθέτησης των αισθητήρων, προσδιορίζοντας την θέση και τον τύπο του αισθητήρα, με απώτερο στόχο την κάλυψη του πεδίου με το ελάχιστο κόστος. Για το λόγο αυτό αναπτύχτικαν δύο μοντέλα για ακριβή και αβέβαιη κάλυψη [5].

Έστω ένα πεδίο N αισθητήρων, οι οποίοι ανήκουν σε ένα πλέγμα δύο ή τριών διαστάσεων. Θεωρούμε K διαφορετικούς τύπου αισθητήρων με εύρος κάλυψης αντίστοιχο του κόστους τους. Η ευκλείδεια απόσταση μεταξύ δυο σημείων (i, j) και (x, y) είναι $d_{ij} = \sqrt{(x-i)^2 + (y-j)^2}$ (33).

2.8 Μοντέλο Ακριβούς Κάλυψης

Στο μοντέλο αυτό θεωρούμε ότι η κάλυψη είναι τέλεια και ένας αισθητήρας καλύπτει πάντα ένα στόχο στο εύρος κάλυψης του. Έστω a_{ijk} ο συντελεστής κάλυψης, με τιμή 1 όταν ο αισθητήρας τύπου k είναι τοποθετημένος στη θέση i και καλύπτει το σημείο j, διαφορετικά 0. Ο αριθμός των συντελεστών κάλυψης σε ένα $N \times N$ δίκτυο με K αισθητήρες είναι KN^2.

Αντικειμενική Συνάρτηση

$$\text{ECP1: } \min \sum_{i=1}^{N} \sum_{k=1}^{K} c_k x_{ik} \qquad (34)$$

$$s.t. \sum_{i=1}^{N} \sum_{k=1}^{K} a_{ijk} x_{ik} \geq b \qquad j = 1,...,N \qquad (35)$$

$$\sum_{k=1}^{K} x_{ik} \leq 1 \qquad i = 1,..,N \qquad (36)$$

$$x_{ik} = 0,1 \qquad i = 1,..,N; k = 1,..,K \qquad (37)$$

Η μεταβλητή x_{ik} έχει τιμή 1 όταν ένας αισθητήρας τύπου k είναι τοποθετημένος στο σημείο i, διαφορετικά 0. Για τον προσδιορισμό της αντικειμενικής συνάρτησης (34 οι περιορισμοί που θέτουμε είναι οι εξής

Κάθε σημείο j να καλύπτεται από τουλάχιστον b αισθητήρες (35) και σε κάθε διαθέσιμο σημείο του grid να μπορεί να τοποθετηθεί ένας και μόνο αισθητήρας.

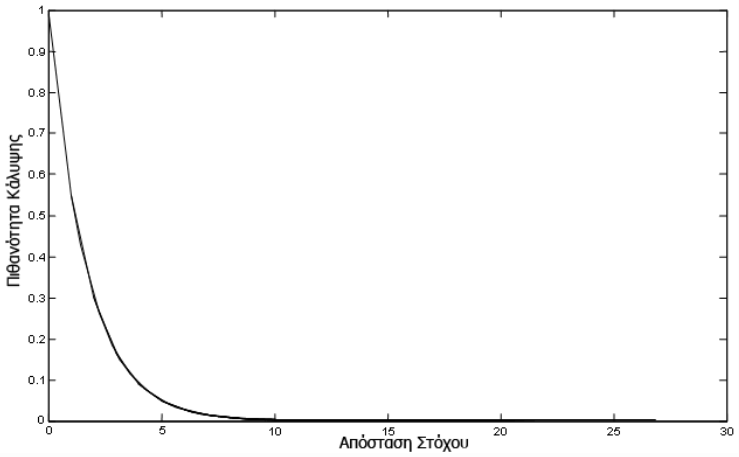
Το κόστος κάθε αισθητήρα είναι εξαρτημένο από τον τύπο του, και προσδιορίζεται από την μεταβλητή c_k. Έτσι η ελαχιστοποίηση της τιμής της αντικειμενικής συνάρτησης, αποτελεί το άθροισμα όλων των τοποθετημένων αισθητήρων στα σημεία του δικτύου.

2.7.2 Μοντέλο Αβέβαιης Κάλυψης

Το μοντέλο ακριβούς κάλυψης, στην πραγματικότητα δεν παρέχει ρεαλιστικά την δυνατή κάλυψη ενός αισθητήρα κάνοντας έτσι την χρήση του περιοριστική στη μοντελοποίηση του δικτύου. Για τον λόγο αυτό, το μοντέλο ακριβούς κάλυψης επεκτείνεται έτσι ώστε να καλύψει την ανάγκη της αβέβαιης κάλυψης του στόχου από ένα αισθητήρα, με χρήση πιθανοτικών μοντέλων.

Έστω p_{ijk} η πιθανότητα κάλυψης ενός σημείου j από έναν αισθητήρα τύπου k τοποθετημένο στη θέση i. Η πιθανότητα κάλυψης ορίζεται εκθετικά με βάση την σχέση e^{-ad} για κάθε i, j, k. Όπου d η ευκλείδεια απόσταση μεταξύ αισθητήρα και στόχου, και a ο συντελεστής προσδιορισμού του ρυθμού μείωσης της πιθανότητας κάλυψης ενός αισθητήρα με βάση την απόσταση. Ο συντελεστής a με τον τρόπο αυτό, εισάγει την ποιότητα του αισθητήρα, και κατά συνέπεια την παράμετρο του κόστους. Έτσι για κάθε τύπο αισθητήρα k η παράμετρος a διαφοροποιείται.

Με βάση την προσέγγιση αυτή, η πιθανότητα κάλυψης μπορεί να γραφεί ως $p_{ijk} = e^{-a_k d_{ij}}$. Το μοντέλο αυτό θεωρεί ότι η απόσταση κάλυψης είναι άπειρη, και μειώνεται ασυμπτωτικά στο μηδέν.



Σχήμα 4 - Πιθανότητα κάλυψης

Η μεταβλητή απόφασης x_{ik} παραμένει ίδια και στο μοντέλο αυτό. Η πιθανότητα να μην βρεθεί ο στόχος στο σημείο j από ένα αισθητήρα τύπου k στο σημείο i είναι $1-p_{ijk}x_{ik}$. Για το σημείο j η συνολική πιθανότητα απώλειας μπορεί να γραφεί ως

$$\prod_{i=1}^{N}\prod_{k=1}^{K}(1-p_{ijk}x_{ik}) \le T \text{ για } 0<T<1 \quad (38)$$

Όπου T η μέγιστη επιτρεπτή πιθανότητα απώλειας. Στην παραπάνω ανισότητα θεωρούμε ότι οι μεταβλητές απόφασης είναι ασυσχέτιστες, και έτσι οι πιθανότητες εντοπισμού στόχου είναι ανεξάρτητες.

Λογαριθμίζοντας και τα δύο μέρη της ανισότητας, η οποία είναι μη γραμμική για δυαδικές τιμές της μεταβλητής x_{ik} έχουμε

$$\sum_{i=1}^{N}\sum_{k=1}^{K}\ln(1-p_{ijk}x_{ik}) \le \ln T \quad (39)$$

Η ανισότητα αυτή εξακολουθεί να είναι μη γραμμική λόγω του λογαρίθμου. Παρόλα αυτά, εφόσον $\ln(1-p_{ijk}x_{ik})=0$ όταν $x_{ik}=0$ και $\ln(1-p_{ijk}x_{ik})=\ln(1-p_{ijk})$ όταν $x_{ik}=1$, η ανισότητα μπορεί να γραφεί ως

$$\sum_{i=1}^{N}\sum_{k=1}^{K}\ln(1-p_{ijk})x_{ik} \le \ln T \quad (40)$$

Θέτοντας $\ln(1-p_{ijk})=a_{ijk}$ και $b=-\ln T$ η παραπάνω ανισότητα μπορεί να μετατραπεί στη μορφή της ανισότητας τέλειας κάλυψης. Οι τιμές αυτές είναι θετικές ως λογάριθμοι τιμών από μηδέν έως ένα. Με τον τρόπο αυτό η αντικειμενική συνάρτηση της αβέβαιης κάλυψης έχει ακριβώς την ίδια μορφή με αυτή της βέβαιης, αλλά με διαφορετικό ορισμό των a_{ijk} και b.

2.7.3 Greedy

Ο υπολογισμός της θέσεως των αισθητήρων, χρησιμοποιεί τον Greedy [17] αλγόριθμο. Αρχικά θεωρούμε ότι όλα τα σημεία είναι διαθέσιμα, και κανένας αισθητήρας δεν έχει τοποθετηθεί. Στο πεδίο των αισθητήρων κάθε σημείο είναι διαθέσιμο εφόσον κανένας αισθητήρας δεν έχει τοποθετηθεί σε αυτό. Θεωρούμε σημείο i^* και αισθητήρα τύπου k^* με τον μεγαλύτερο συντελεστή κέρδους, και τοποθετούμε στο σημείο αυτό i^* τον αισθητήρα τύπου k^*. Το σημείο αυτό είναι πλέον μη διαθέσιμο. Η διαδικασία αυτή ακολουθείται έως ότου όλα τα σημεία j να είναι καλυμμένα. Ένα σημείο είναι καλυμμένο εφόσον είναι καλυμμένο από τουλάχιστον b αισθητήρες.

Αλγόριθμος

1. Αρχικά κάθε σημείο i έχει μηδενική κάλυψη, και $x_{ik} = 0$ για κάθε i, k
2. Εφόσον υπάρχουν διαθέσιμα σημεία i επαναλαμβάνουμε τα βήματα 3-5, διαφορετικά ο αλγόριθμος σταματάει (Διαθέσιμο θεωρείται ένα σημείο όταν δεν καλύπτεται από τουλάχιστον b αισθητήρες, σύμφωνα με τον περιορισμό).
3. Υπολογίζουμε τον συντελεστή κέρδους για κάθε διαθέσιμο σημείο i και τύπο αισθητήρα k
4. Έστω i^* το σημείο με το μέγιστο κέρδος για αισθητήρα k^*.
5. Τοποθετούμε τον αισθητήρα στο σημείο αυτό θέτοντας $x_{ik} = 1$, και ενημερώνοντας την κάλυψη των διαθέσιμων σημείων.
6. Αν υπάρχουν ακόμα διαθέσιμα σημεία μεταβαίνουμε στο σημείο 2 του αλγορίθμου, διαφορετικά αλγόριθμος σταματάει μιας και κάθε σημείο καλύπτεται πλέον από τουλάχιστον b σημεία

Το χειρότερο σενάριο για την κάλυψη της περιοχής, θα ήταν να χρησιμοποιηθούν μόνο οι αισθητήρες με την μεγαλύτερη κάλυψη σε κάθε σημείο. Αυτό όμως δεν είναι επιθυμητό για λόγους κόστους. Ο συντελεστής κέρδους συνδυάζει την εφαρμογή των

περιορισμών της αντικειμενικής συνάρτησης με το κόστος των διαφορετικών αισθητήρων και προσδιορίζεται διαφορετικά για κάθε μοντέλο κάλυψης.

Για ακριβή εντοπισμό, θεωρούμε ότι η κάλυψη ενός σημείου j, η οποία συμβολίζεται ως $\text{cov}(j)$, και ορίζεται από τον αριθμό των αισθητήρων στο εύρος κάλυψης του σημείου j. Τότε, η ποσότητα max(0, b-cov(j)) μπορεί να οριστεί ως η μη κάλυψη του σημείου j.

Χρησιμοποιώντας την στρατηγική κάλυψης, με τιμές 0 ή 1 των μεταβλητών x_{ik} η κάλυψη ενός σημείου j δίνεται από την σχέση:

$$\text{cov}(j) = \sum_{i=1}^{N}\sum_{k=1}^{K} a_{ijk} x_{ik} \quad (41)$$

Ο συντελεστής κέρδους για το σημείο i και αισθητήρα τύπου k εκφράζεται ως

$$\gamma_{ik} = \frac{\sum_{j=1}^{N} a_{ijk} \max(0, b - \text{cov}(j))}{c_k} \quad (42)$$

Ο αριθμητής καθορίζει τη μη κάλυψη στα σημεία που μπορούν να καλυφθούν από ένα αισθητήρα k τοποθετημένο στο σημείο i, και ο παρανομαστής είναι το κόστος του αισθητήρα τύπου k. Έτσι ο συντελεστής κέρδους υπολογίζει την δυνατή μείωση ανά μονάδα κόστους όταν ένας αισθητήρας τύπου k τοποθετηθεί στο σημείο i.

Ο συντελεστής κέρδους για την περίπτωση αβέβαιης κάλυψης διαφοροποιείται μιας και ο συντελεστής a_{ijk} είναι συσχετισμένος με την πιθανότητα κάλυψης. Η πιθανότητα απώλειας του σημείου j, $miss(j)$, ορίζεται ως

$$miss(j) = \prod_{i=1}^{N} \prod_{k=1}^{K} (1 - p_{ijk}) \quad (43)$$

Η μη κάλυψη ενός σημείου j ορίζεται από την ποσότητα $\max(0, miss(j) - T)$, και ο συντελεστής κέρδους ορίζεται ως

$$\gamma_{ik} = \frac{\sum_{j=1}^{N} p_{ijk} \max(0, miss(j) - T)}{c_k} \quad (44)$$

2.7.4 Lagrangean

Η τεχνική Lagrangean relaxation, είναι σημαντική για την επίλυση προβλημάτων ακέραιου προγραμματισμού και συνδυαστικών προβλημάτων βελτιστοποίησης, και μπορεί να χρησιμοποιηθεί για την ανάπτυξη ενός ακριβή και αποδοτικού αλγορίθμου Lagrangean. Για την ανάπτυξη του αλγορίθμου χρησιμοποιούμε το ακόλουθο πρόβλημα.

$$\text{BIP: } z^* = \min \sum_{j=1}^{n} c_j x_j \quad (45)$$

$$s.t. \sum_{j=1}^{n} a_{ij} x_j \geq b_i \qquad i = 1, ..., m_1 \quad (46)$$

$$\sum_{j=1}^{n} d_{ij} x_j \leq d_i \qquad i = 1, ..., m_2 \quad (47)$$

$$x_j = 0,1 \qquad j = 1, ..., n \quad (48)$$

Η μεταβλητή x_j είναι η μεταβλητή απόφασης. Ορίζουμε το Lagrangean relaxation του προβλήματος εισάγοντας θετικούς Lagrange πολλαπλασιαστές $\mu = (\mu_i \ : \ i = 1, ..., m_1)$ τους οποίους και εισάγουμε στην αντικειμενική συνάρτηση για καθορισμό του Lagrangean υπό-προβλήματος.

$$LR(\mu): z^*_{LR}(\mu) = \min \sum_{j=1}^{n} c_j x_j + \sum_{i=1}^{m_1} \mu_i \left(bi - \sum_{j=1}^{n} a_{ij} x_j \right) \quad (49)$$

$$s.t \sum_{j=1}^{n} d_{ij} x_j \geq d_i \quad i = 1,...,m2 \quad (50)$$

$$x_j = 0,1 \quad \text{j=1,...,n} \quad (51)$$

Η βέλτιστη αντικειμενική τιμή $z^*_{LR}(\mu)$ είναι ένα κατώτερο όριο της αντικειμενικής τιμής z^* του αρχικού προβλήματος. Υπάρχουν δυο θέματα τα οποία σχετίζονται με την με την απλοποίηση (relaxation) Lagrangean.

1. Ποιο περιορισμοί θα πρέπει να ελαχιστοποιηθούν;
2. Πως θα πρέπει να επιλεγούν οι αριθμητικές τιμές των πολλαπλασιαστών έτσι ώστε να μεγιστοποιηθεί το κατώτερο όριο;

Η απάντηση στο πρώτο ερώτημα θα επηρεάσει την μορφή του υπό-προβλήματος και την επίλυση του. Στο δεύτερο η λύση προκύπτει από επαναληπτική μέθοδο η οποία αποτελείτε από πολλές λύσεις του Lagrangean υπό-προβλήματος , αναζητώντας τους καλύτερους συντελεστές που θα δώσουν την καλύτερη λύση στο αρχικό πρόβλημα.

Οι καλύτερες πιθανές τιμές των πολλαπλασιαστών προκύπτουν από την επίλυση του Lagrangean διπλού προβλήματος.

$$z^*_{LR}(\mu^*) = \max_{\mu \geq 0} \{z^*_{LR}(\mu)\} \quad (52)$$

Η ιδανική αντικειμενική τιμή του διπλού προβλήματος $z^*_{LR}(\mu^*)$ είναι ίση με την ιδανική z^* του αρχικού προβλήματος.

Η βασική ιδέα είναι η δημιουργία μιας ακολουθίας από πολλαπλασιαστές Lagrangean οι οποίοι θα παράγουν τα κατώτερα όρια, και μια ακολουθία από δυνατές λύσεις οι οποίες

θα παράγουν τα ανώτατα όρια του αρχικού προβλήματος. Όταν ολοκληρωθεί η διαδικασία αυτή η καλύτερη λύση είναι και η λύση του προβλήματος.

Ελαχιστοποιώντας και τους περιορισμούς (2) και (3) ορίζουμε το υπό - πρόβλημα.

$$LR(\mu): z^*_{LR}(\mu) = \min \sum_{i=1}^{N}\sum_{k=1}^{K} c_k x_{ik} + \sum_{j=1}^{N} \mu_j \left(b - \sum_{i=1}^{N}\sum_{k=1}^{K} a_{ijk} x_{ik} \right) + \sum_{i=1}^{N} \nu_i \left(\sum_{k=1}^{K} x_{ik} - 1 \right)$$
$$s.t \qquad x_{ik} = 0,1 \qquad i = 1,..,N; k = 1,..,K$$

Ισοδύναμα η αντικειμενική συνάρτηση μπορεί να γραφτεί

$$\min \sum_{i=1}^{N}\sum_{k=1}^{K} \left(c_k + \nu_i - \sum_{j=1}^{N} \mu_j a_{ijk} \right) x_{ik} + \sum_{j=1}^{N} \mu_j b - \sum_{i=1}^{N} \nu_i \quad (53)$$

Αρχικά εφαρμόζουμε τον Greedy αλγόριθμο για να πάρουμε μια αρχική λύση (άνω όριο z^*) , Στην συνέχεια εφαρμόζοντας τον Lagrangean Αλγόριθμο βρίσκουμε την ελάχιστη από τις τιμές της αντικειμενικής συνάρτησης η οποία προκύπτει.

1. Αρχικοποίηση
    1. Εφαρμογή του Greedy αλγορίθμου για την εύρεση της αρχικής δυνατής λύσης.
    2. Θέτουμε το άνω όριο σε $z_{UB} = z_G$, και το κατώτερο όριο σε $z_{LB} = -\infty$
    3. Θέτουμε $\mu_j^{(0)} = 1$, $\nu_j^{(0)} = 1$ και $t = 0$
2. Κύριο Βήμα
    1. Επανάληψη για προκαθορισμένο αριθμό επαναλήψεων
    2. Επίλυση του υπό προβλήματος $LR(\mu^{(t)}, \nu^{(t)})$ με έλεγχο των τιμών των $\mu^{(t)}$ και $\nu^{(t)}$ για να πετύχουμε τον μετριασμό της λύσης και να υπολογίσουμε το $z_{LR}(\mu^{(t)}, \nu^{(t)})$

3. Εάν $z_{LR}(\mu^{(t)}, v^{(t)}) > z_{LB}$ τότε θέτουμε $z_{LB} = z^*_{LR}(\mu^{(t)}, v^{(t)})$, διαφορετικά το z_{LB} παραμένει ίδιο.
4. Εάν η λύση είναι εφαρμόσιμη για το αρχικό πρόβλημα
5. Υπολογίζουμε την αντικειμενική λύση $\bar{z}$, και πηγαίνουμε στο βήμα 2.7
6. διαφορετικά χρησιμοποιούμε την τιμή από τον Greedy αλγόριθμο $\bar{z} = z_G$
7. Εάν $\bar{z} < z_{UB}$
8. τότε $z_{UB} = \bar{z}$ διαφορετικά το z_{UB} παραμένει αμετάβλητο.
9. Ενημέρωση των πολλαπλασιαστών

$$\mu_j^{(t+1)} = \mu_j^{(t)} + \eta^{(t)} \left(b - \sum_{i=1}^{N} \sum_{k=1}^{K} a_{ijk} x_{ik} \right)$$

$$v_i^{(t+1)} = v_i^{(t)} + \delta^{(t)} \left(\sum_{k=1}^{K} x_{ik} - 1 \right)$$

$$\eta^{(t)} = \frac{\theta \left(z_{UB} - z^*_{LR}(\mu_j^{(t)}, v_j^{(t)}) \right)}{\sum_{j=1}^{N=1} \left(b - \sum_{i=1}^{N} \sum_{k=1}^{K} a_{ijk} x_{ik} \right)^2}$$

$$\delta^{(t)} = \frac{\theta (z_{UB} - z^*_{LR}(\mu_j^{(t)}, v_j^{(t)}))}{\sum_{k=1}^{N} \left(\sum_{k=1}^{K} x_{ik} - 1 \right)^2}$$

Αρχικά οι πολλαπλασιαστές έχουν τιμή 1 και $\theta = 2$.

3. Μεθοδολογία

3.1 Εισαγωγή

Οι μέθοδοι τοποθέτησης αισθητήρων που αναλύθηκαν στην βιβλιογραφική επισκόπηση, είτε θεωρούν ότι η κάλυψη είναι τέλεια είτε προσπαθούν να αποδώσουν τις απώλειες λόγων των περιβαλλοντολογικών συνθηκών θεωρώντας ότι η κάλυψη μειώνεται εκθετικά με την απόσταση. Η δυνατότητα εντοπισμού ενός στόχου από έναν αισθητήρα, εξαρτάται από τους περιβαλλοντολογικούς παράγοντες σε πραγματικές συνθήκες τοποθέτησης, συνεπώς επηρεάζει την σχεδίαση τοποθέτησης [2]. Για το λόγο αυτό θα πρέπει στην σχεδίαση του αλγορίθμου τοποθέτησης να ληφθεί υπόψη και το φαινόμενο σκίασης.

3.2 Επιλογή Αλγορίθμου επίλυσης

Για την ανάπτυξη του αλγορίθμου επιλέγουμε να επεκτείνουμε το μοντέλο γραμμικού – ακέραιου προγραμματισμού [5]. Επιλέγουμε την προσέγγιση αυτή, γιατί ο αλγόριθμός αυτός έχει σχεδιαστεί έτσι ώστε να προσαρμόζεται στο μοντέλο κάλυψης ανεξάρτητα από το πως ορίζεται αυτό, στηρίζοντας την λύση του σε δεδομένες τεχνικές επίλυσης όπως ο αλγόριθμός Greedy.

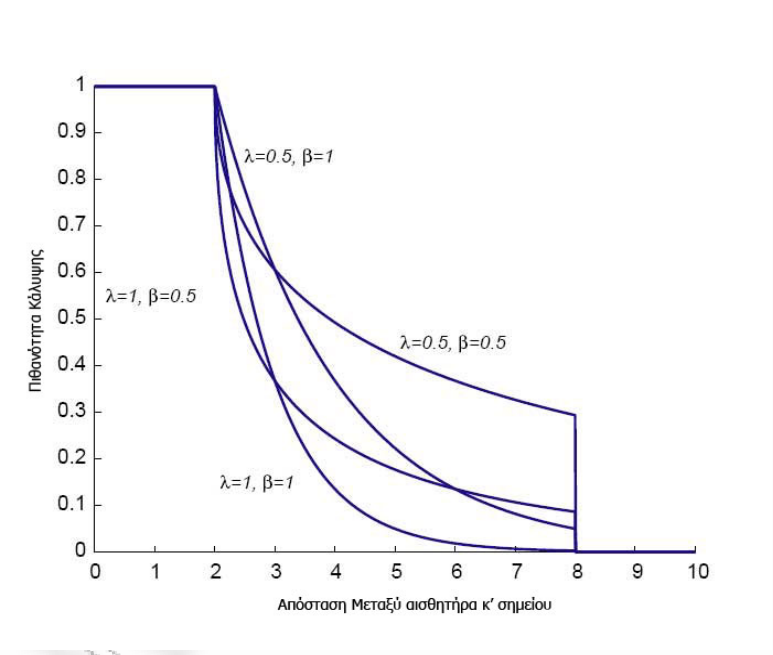
3.3 Μοντέλο αβέβαιης κάλυψης για περιορισμένη ακτίνα κάλυψης

Στο μοντέλο αβέβαιης κάλυψης [5], θεωρεί ότι το εύρος κάλυψης του κάθε αισθητήρα είναι άπειρο και μειώνεται ασυμπτωτικά στο μηδέν. Για τον καλύτερο προσδιορισμό του μοντέλου υπολογίζουμε την πιθανότητα κάλυψης, λαμβάνοντας υπόψη την περίπτωση οπού ο στόχος είναι σε απόσταση μεγαλύτερη από αυτή της κάλυψης του αισθητήρα [11].

$$p_{ijk} = \begin{cases} 0 & if \quad r + r_e \le d_{ij}(x,y) \\ e^{-\lambda\alpha^{\beta}} & if \quad r - r_e \prec d_{ij}(x,y) \prec r + r_e \\ 1 & if \quad r - r_e \ge d_{ij}(x,y) \end{cases} \quad (54)$$

Όπου $r_e (r_e < r)$ είναι με μέτρο αβεβαιότητας εντοπισμού του στόχου από τον αισθητήρα τύπου k. Η παράμετρος αυτή εισάγει την απόσταση εκείνη κάτω από την οποία η κάλυψη του στόχου είναι βέβαιη. Οι παράμετροι $a = d_{ij}(x,y) - (r - r_e)$ και λ, β είναι παράμετροι υπολογισμού της πιθανότητας κάλυψης όταν ένας αισθητήρας είναι σε απόσταση μεγαλύτερη από r_e, αλλά μέσα στην ακτίνα κάλυψης του αισθητήρα.

Οι αποστάσεις όπως και στα προηγούμενα δυο μοντέλα μετρούνται με μονάδες σημείων του πλέγματος. Το μοντέλο παρουσιάζεται στο παρακάτω Σχήμα.



Σχήμα 5 – Πιθανότητα Κάλυψης με περιορισμένο εύρος.

Το διάγραμμα παρουσιάζει επίσης την επίδραση της απόστασης στην κάλυψη από τον αισθητήρα έως το σημείο εμπιστοσύνης. Διαφορετικές τιμές των συντελεστών λ και β

προσδιορίζουν τους διαφορετικούς τύπους αισθητήρων k, κατά συνέπεια και την διαφορετική δυνατότητα κάλυψης.

Τα παραπάνω μοντέλα, αν και λαμβάνουν υπόψη τους την αβεβαιότητα κάλυψης των αισθητήρων χρησιμοποιώντας πιθανοτικά μοντέλα κάλυψης, δεν χρησιμοποιούν ρεαλιστικά μοντέλα κάλυψης. Για τον προσδιορισμό των πραγματικών χαρακτηριστικών κάλυψης, εισάγουμε στο υπάρχον μοντέλο τις απώλειες διάδοσης σήματος.

Οι βασικοί τύποι διαλείψεων που πρέπει να ληφθούν υπόψη είναι οι αργές διαλείψεις και οι διαλείψεις με σκίαση. Οι διαλείψεις αυτές προκαλούνται από εμπόδια στην διαδρομή διάδοσης μεταξύ του σταθμού βάσης και του κινητού.

3.4 Μοντέλα διάδοσης μεγάλης κλίμακας

Η στιγμιαία λαμβανόμενη ισχύς εμφανίζει απότομες διακυμάνσεις καθώς το κινητό διανύει μικρές αποστάσεις (μικρά μήκη κύματος). Τα μοντέλα διάδοσης μεγάλης κλίμακας προβλέπουν την μεταβολή της μέσης ισχύος του σήματος ως συνάρτηση της απόστασης από τον πομπό. Έτσι μπορούν να χρησιμοποιηθούν για την εκτίμηση της περιοχής κάλυψης των ασύρματων αισθητήρων.

3.4.1 Μοντέλο διάδοσης ελεύθερου χώρου

Η ισχύς σε μια κεραία λήψης σε απόσταση d από την κεραία εκπομπής στον ελεύθερο χώρο σύμφωνα με την εξίσωση Friis είναι

$$\Pr(d) = Pt\frac{GtGr\lambda^2}{(4\pi)^2 d^2} \quad (55)$$

όπου Pt είναι η εκπεμπόμενη ισχύς, Pr(d) η λαμβανόμενη ισχύς που είναι συνάρτηση της απόστασης, Gt το κέρδος της κεραίας του πομπού, Gr το κέρδος της κεραίας του δέκτη, d η απόσταση πομπού δέκτη σε μέτρα και λ είναι το μήκος κύματος σε μέτρα.

Η απώλεια διαδρομής (path loss), η οποία αντιπροσωπεύει τη εξασθένιση του σήματος σε dB, ορίζεται ως η διαφορά ανάμεσα στην ενεργό εκπεμπόμενη ισχύ και τη λαμβανόμενη. Άρα η απώλεια διαδρομής για το μοντέλο ελεύθερου είναι:

$$PL_{DB} = 10\log\frac{Pt}{\Pr} = -10\log\frac{GtGr\lambda^2}{(4\pi)^2 d^2} \quad (56)$$

Το μοντέλο αυτό είναι έγκυρο για τιμές του d που αντιστοιχούν στο μακρινό πεδίο της κεραίας του πομπού. Επιπλέον, είναι προφανές ότι η εξίσωση δεν ισχύει για d=0. Για το λόγο αυτό τα μοντέλα μεγάλης κλίμακας χρησιμοποιούν μια κοντινή απόσταση d_0 ως σημείο αναφοράς, οπότε το μοντέλο απωλειών στον ελεύθερο χώρο εκφράζεται με τη μορφή

$$\Pr(d) = \Pr(d_0)\left(\frac{d}{d_0}\right)^n \quad (57)$$

Με μέση τιμή απωλειών

$$\overline{PL}(d) = \overline{PL}(d_0) + 10n\log_{10}\left(\frac{d}{d_0}\right) \quad (58)$$

Η τιμή του εκθέτη απωλειών n εξαρτάται από το περιβάλλον διάδοσης, και καθορίζει το ρυθμό με τον οποίο οι απώλειες διαδρομής αυξάνουν με την απόσταση

3.4.2 Long –normal schadowing

Το απλοποιημένο μοντέλο απωλειών διαδρομής προβλέπει τις ίδιες απώλειες για όλες τις διαδρομές με το ίδιο μήκος d. Στην πράξη όμως οι τιμές αυτές μπορεί να διαφέρουν σημαντικά από τις πραγματικές λόγω της διαφορετικής συγκέντρωσης εμποδίων που ακολουθεί το σήμα. Η επίδραση της σκίασης λαμβάνεται υπόψη στο ακόλουθο στοχαστικό μοντέλο

$$\overline{L}(d) = \overline{L}(d_0) + 10n\log_{10}\left(\frac{d}{d_0}\right) + X\sigma \quad (59)$$

Όπου $X\sigma$ είναι τυχαία μεταβλητή Gauss (σε dB) με μηδενική μέση τιμή και τυπική απόκλιση ίση με σ (σε dB).

Σε συγκεκριμένη απόσταση d, η πιθανότητα η λαμβανόμενη ισχύς να είναι μικρότερη από ένα συγκεκριμένο κατώφλι γ είναι

$$P(\Pr(d) < \gamma) \quad (60)$$

Η σχέση αυτή μπορεί να γραφτεί με βάση το μοντέλο απωλειών διάδοσης

$$\mathrm{P}(\mathrm{L(d)} > \beta) \quad (61)$$

Όπου β η μέγιστη ανεκτή απώλεια διάδοσης

Εφόσον L(d) είναι τυχαία μεταβλητή Gauss

$$\mathrm{P}(\mathrm{X} > b) = \mathrm{Q}\left(\frac{\mathrm{b - a}}{\sigma}\right) \quad (62)$$

Όπου Q είναι η συνάρτηση η οποία ορίζεται ως

$$\mathrm{Q(z)} = \frac{1}{\sqrt{2\pi}} \int_{z}^{+\infty} \exp\left(-\frac{x^2}{2} \right) dx \quad (63)$$

$$\mathrm{Q(z)} = 1 - \mathrm{Q(-z)} \quad (64)$$

Άρα η πιθανότητα κάλυψης ενός σημείου σε απόσταση d είναι

$$\Pr[\Pr(d) > b] = \mathrm{Q}\left(\frac{\mathrm{b - Pr(d)}}{\sigma} \right) \quad (65)$$

3.4.3 Χωρική Συσχέτιση

Οι τιμές των απωλειών σκίασης δεν θεωρούνται απαραίτητα συσχετισμένες. Στην πραγματικότητα όμως, θέματα συσχέτισης προκύπτουν στις περισσότερες περιπτώσεις αργών διαλείψεων. Το φαινόμενο των χωρικών συσχετίσεων, έχει αποδειχτεί πολύ σημαντικό για την απόδοση των ασύρματων αισθητήρων [13].

Οι απώλειες διαδρομών από ένα πομπό στη θέση l^t και δυο δέκτες στις θέσεις l^a και l^b είναι

$$PL(l^b) = 10n \log(\frac{4\pi d(l^b, l^t)}{\lambda}) + \chi_{\sigma^2}(l^b) \quad (66)$$

$$PL(l^a) = 10n \log(\frac{4\pi d(l^a, l^t)}{\lambda}) + \chi_{\sigma^2}(l^a) \quad (67)$$

Όπου $d(l^x, l^y)$ η απόσταση μεταξύ των δυο δεκτών, $\chi_{\sigma^2}(l^b)$, $\chi_{\sigma^2}(l^a)$ οι log-normal κατανομές με μηδενική μέση τιμή και τυπική απόκλιση ίση με σ (σε dB), n ο βαθμός απωλειών και λ το μήκος κύματος του σήματος.

Η δημιουργία συσχετισμένων Gaussian μεταβλητών χ_{A1}, χ_{A2} βασίζεται στις ακόλουθες σχέσεις

$$\chi_{A1} = \sigma_{A1}\left(\sqrt{|\rho|}\chi_{B1} + \sqrt{1-|\rho|^2}\chi_{B2}\right) + \mu_{A1} \quad (68)$$

$$\chi_{A2} = \sigma_{A2}\left(\sqrt{|\rho|}\chi_{B1} + \sqrt{1-|\rho|^2}\chi_{B3}\right) + \mu_{A2} \quad (69)$$

Όπου σ_{A1}, σ_{A2} είναι οι τυπικές αποκλίσεις και μ_{A1}, μ_{A2} οι μέσες τιμές των παραγόμενων δειγμάτων. $\chi_{B1}, \chi_{B2}, \chi_{B3}$ είναι οι ασυσχέτιστες Gaussian μεταβλητές με ρ τον συντελεστή συσχέτισης ο οποίος δίνετε από την σχέση:

$$\rho(l^b, l^a) = \sigma^2 \exp\left(-\frac{d(l^b, l^a)}{d^*}\right) \quad (70)$$

Ο παράγοντας d^* καθορίζει την κρίσιμη απόσταση συσχέτισης η οποία εξαρτάται από το χώρο και τον τύπο του αισθητήρα και $\sigma^2 = \sigma_{A1} * \sigma_{A2}$ (71)

Για την δημιουργία N συσχετισμένων Gaussian δειγμάτων, χρησιμοποιείται η ακόλουθη σχέση

$$X_A = TX_B + M \quad (72)$$

Όπου τ ο κατώτερος τριγωνικός πίνακας που προέρχεται από τον covariance πίνακα σ με βασική (Choleski) decomposition, μ το διάνυσμα των μέσων τιμών και x_B το σύνολο των ανεξάρτητων, μηδενικής μέσης τιμής Gaussian ακολουθίας.

4. Αποτελέσματα

4.1 Εισαγωγή

Η εκτέλεση των αλγόριθμων έχει πραγματοποιηθεί σε υπολογιστή με Intel(R) Core(TM)2 επεξεργαστή στα 2,16GHz, με 2,00 GB RAM και λειτουργικό σύστημα Microsoft Windows Server 2003 R2.

Η ανάπτυξη των αλγορίθμων έχει πραγματοποιηθεί με χρήση τριών γλωσσών προγραμματισμού οι οποίες είναι

- Matlab, έκδοση 7.3.0.267 (R2006b),
- Microsoft .NET framework έκδοση 2.0.50727, με χρήση Microsoft Visual Studio 2005 8.0.50727.43
- yaBasic

Για τα προβλήματα ακέραιου και γραμμικού προγραμματισμού, έχει χρησιμοποιηθεί LINGO Extended 10.00 Educational Windows [18], [19]. Όλα τα προβλήματα ακέραιου προγραμματισμού έχουν χρόνο εκτέλεσης 1 ώρα.

4.2 Τέλεια Κάλυψη

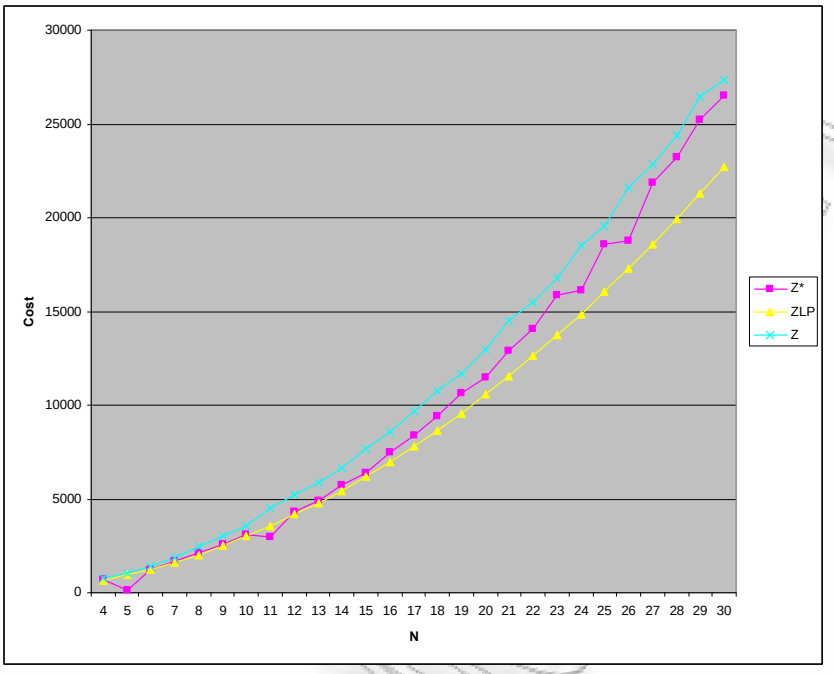
Οι δοκιμές των αλγορίθμων έχουν πραγματοποιηθεί με χρήση δυο τύπων αισθητήρων με εύρος 1 και 2 σημεία αντίστοιχα, και κόστος 100 και 150. Υποθέτουμε ότι το πεδίο των αισθητήρων είναι σε ένα πλέγμα δύο διαστάσεων, για 36 διαφορετικά $N = n \times n$ πλέγματα, με $n \in \{4,5,...,40,50\}$. Κάθε δοκιμή έχει $n^2 \times n^2 \times K$ δυαδικές μεταβλητές, με εύρος από $16 \times 16 \times 1 = 256$ έως $2500 \times 2500 \times 2 = 12500000$. Επιλέγουμε $b = 2$.

Αρχικά επιλύουμε, τα προβλήματα γραμμικού και ακέραιου προγραμματισμού με χρήση του Lingo. Επίσης, για τις ίδιες διαστάσεις, επιλύουμε το πρόβλημα με τον Greedy αλγόριθμο. Συμβολίζουμε με z_G την αντικειμενική τιμή που υπολογίζετε με τον Greedy, με z_{LP} την αντικειμενική τιμή του γραμμικού προγραμματισμού και z^* την λύση από το πρόβλημα του ακέραιου προγραμματισμού.

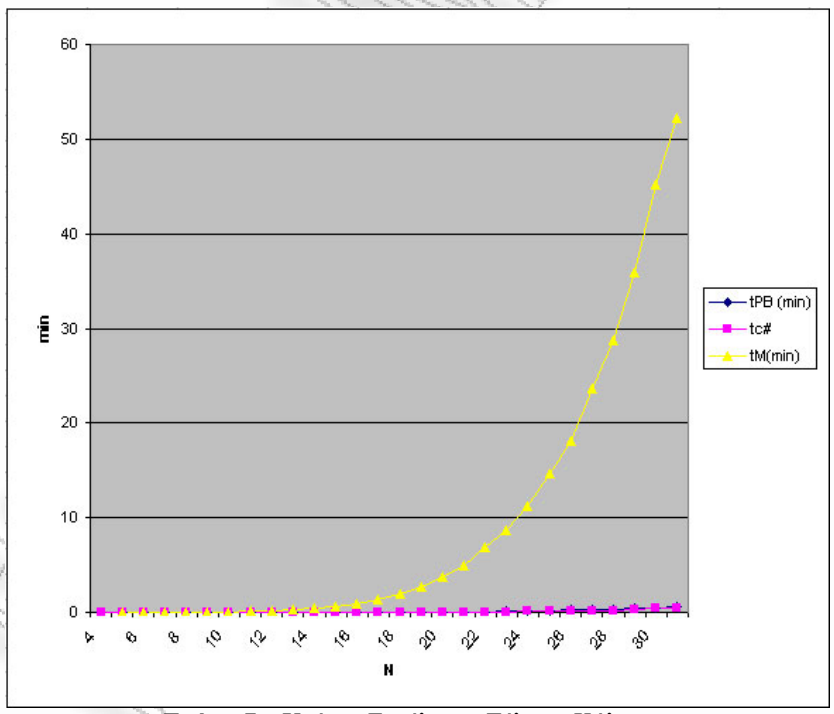
Για το πρόβλημα γραμμικού προβληματισμού, αντικαθιστούμε τον περιορισμό για δυαδική τιμή του x_{ik}, με τα όρια $0 \le x_{ik} \le 1$. Έτσι z_{LP} είναι η κάτω όριο του z^*.

n	Z*	Z_{LP}	Z_G	$n_{k=1}$	$n_{k=2}$	n_{TOT}	t_{PB} (sec)	t_{PB} (min)	$t_{c\#}$ (sec)	$t_{c\#}$ (min)	t_M (sec)	t_M(min)
4	700	666.7	800	2	4	6	0.00	0.00	0.00	0.00	0.07	0.00
5	1000	960	1050	0	7	7	0.00	0.00	0.00	0.00	0.10	0.00
6	1200	1200	1400	5	6	11	0.36	0.01	0.00	0.00	0.32	0.01
7	1700	1645	1900	4	10	14	0.05	0.00	0.01	0.00	0.70	0.01
8	2100	2000	2450	5	13	18	0.27	0.00	0.01	0.00	1.52	0.03
9	2550	2513	3050	8	15	23	0.00	0.00	0.02	0.00	3.07	0.05
10	3100	3015	3550	7	19	26	0.34	0.01	0.04	0.00	5.32	0.09
11	3000	3524	4500	12	22	34	0.14	0.00	0.08	0.00	9.83	0.16
12	4300	4169	5200	16	24	40	0.09	0.00	0.13	0.00	16.25	0.27
13	4900	4773	5900	11	32	43	0.14	0.00	0.19	0.00	24.36	0.41
14	5750	5450	6650	17	33	50	0.03	0.00	0.30	0.00	38.02	0.63
15	6400	6215	7650	12	43	55	0.48	0.01	0.45	0.00	55.53	0.93
16	7500	6959	8600	17	46	63	0.89	0.01	0.65	0.00	81.84	1.36
17	8400	7835	9650	17	53	70	0.55	0.01	0.95	0.00	115.48	1.92
18	9450	8673	10800	21	58	79	1.44	0.02	1.37	0,01	164.36	2.74
19	10650	9550	11700	21	64	85	1.72	0.03	1.84	0,01	220.91	3.68
20	11500	10580	12950	23	71	94	2.28	0.04	2.46	0,02	298.48	4.97
21	12900	11554	14500	25	80	105	3.39	0.06	2.83	0.05	409.27	6.82
22	14050	12621	15500	20	90	110	6.28	0.10	3.22	0.05	519.32	8.66
23	15900	13726	16800	24	96	120	8.62	0.14	4.55	0.08	673.91	11.23
24	16150	14847	18520	31	101	132	11.91	0.20	6.24	0.10	875.62	14.59
25	18600	16088	19550	23	115	138	15.03	0.25	7.04	0.12	1082.72	18.05
26	18750	17305	21600	33	122	155	19.23	0.32	9.81	0.16	1414.04	23.57
27	21900	18577	22850	32	131	163	20.39	0.34	12.89	0.21	1722.81	28.71
28	23250	19940	24400	37	138	175	25.14	0.42	18.15	0.30	2151.22	35.85
29	25250	21301	26450	50	143	193	31.11	0.52	23.31	0.39	2714.60	45.24
30	26500	22738	27350	29	163	192	37.03	0.62	26.21	0.44	3135.23	52.25

Πίνακας 1 – Αποτελέσματα Τέλειας Κάλυψης



Σχήμα 6 – Κόστος Κάλυψης Τέλειας Κάλυψης



Σχήμα 7 – Χρόνος Εκτέλεσης Τέλειας Κάλυψης

Από τα αποτελέσματα της εκτέλεσης του αλγορίθμου, παρατηρούμε ότι η τιμή της αντικειμενικής συνάρτησης αυξάνεται ομοιόμορφα και με τους τρεις τρόπους υπολογισμού της (Ακέραιος, γραμμικός προγραμματισμός και Greedy). Η διαφοροποίηση οφείλεται στο γεγονός ότι ο περιορισμός $x_{ik} = 0,1$ παύει να ισχύει κατά την επίλυση του LP προβλήματος μιας και το εύρος τιμών του x_{ik} είναι μεταξύ 0 και 1. Επίσης η επίλυση του IP προβλήματος διακόπτεται στην 1 ώρα, οπότε τα αποτελέσματα από την επίλυση είναι προσεγγιστικά.

Οι αισθητήρες μεγαλύτερου κόστους (κ=2) είναι περισσότεροι σε σχέση με το πλήθος των αισθητήρων μικρότερου κόστους (κ=1) για κάθε μέγεθος του δικτύου. Η διαφορά αυτή αυξάνεται όσο αυξάνετε και το μέγεθος του δικτύου.

Σχετικά με τον χρόνο εκτέλεσης παρατηρούμε ότι αυξάνεται εκθετικά. Είναι ιδιαίτερα σημαντική η διαφορά στον χρόνο εκτέλεσης που απαιτείται από το Matlab σε σχέση με τις γλώσσες τρίτης γενιάς όπως η C# και η yaBasic, ειδικά όσο το μέγεθος του δικτύου αυξάνει.

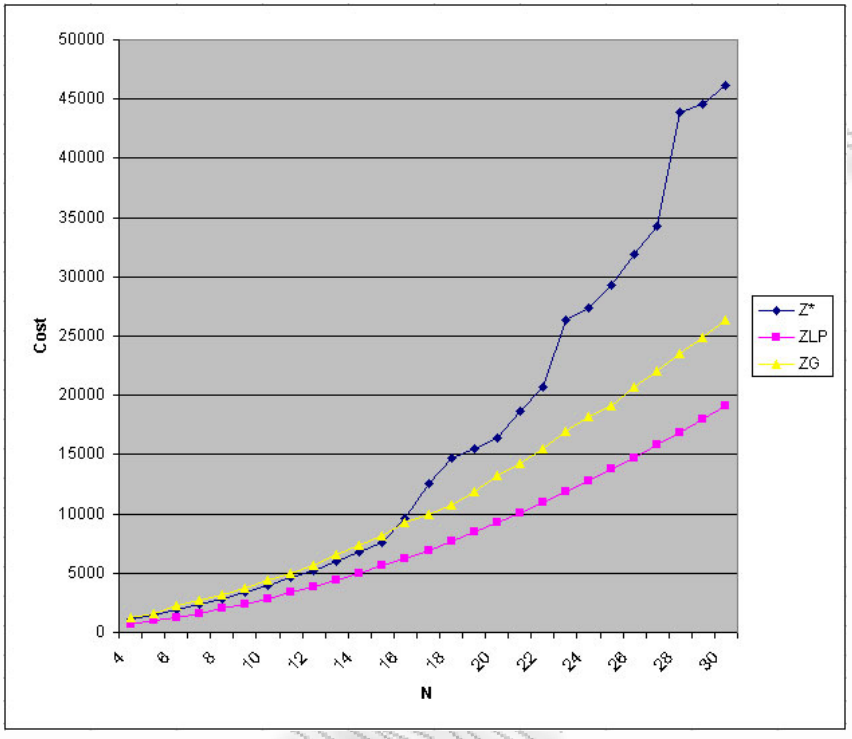
Θα πρέπει να τονιστεί πως η υπολογιστική ισχύς που απαιτείται, όπως και ο χρόνος εκτέλεσης είναι σημαντικοί παράγοντες επιλογής στον τρόπο υλοποίησης των αλγορίθμων. Με χρήση του Matlab όπως φαίνεται και στα αποτελέσματα επίλυσης, αν και έχουμε άμεσο έλεγχο της λειτουργίας του αλγορίθμου με ελάχιστο κώδικά, δεν μπορούμε να προχωρήσουμε σε λύση για μεγάλα δίκτυα αισθητήρων, μιας και ο χρόνος που απαιτείται είναι τεράστιος σε σχέση με αυτόν που απαιτείται από μια γλώσσα τρίτης γενιάς. Έτσι για πραγματικές συνθήκες κάλυψης μεγάλων δικτύων, η χρήση των υλοποιημένων αλγορίθμων από μια γλώσσα τρίτης γενιάς κρίνεται απαραίτητη.

4.3 Αβέβαιη Κάλυψη

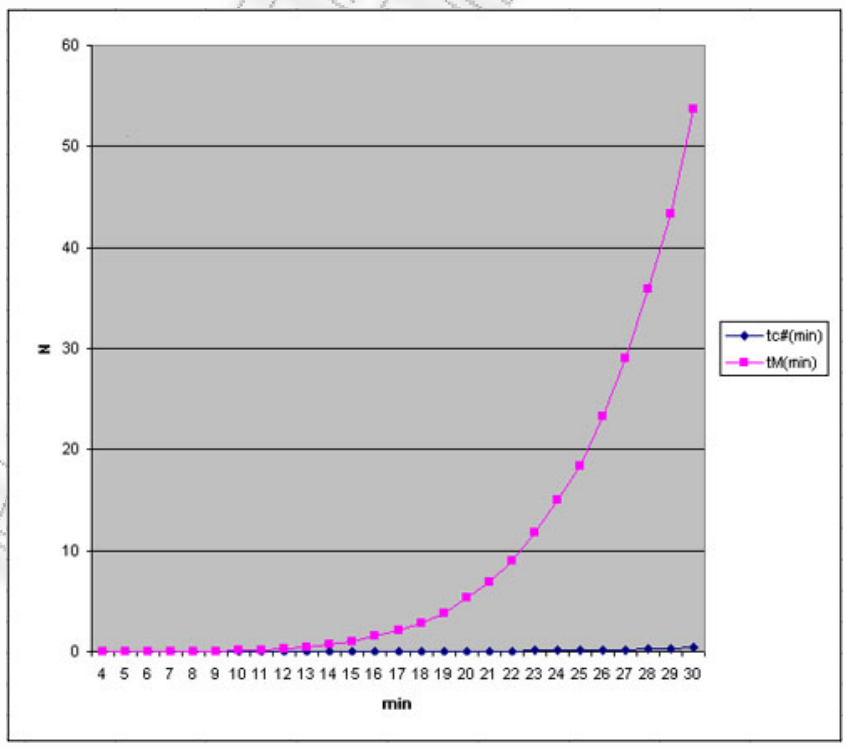
Θεωρούμε ομοίως δυο τύπους αισθητήρων συντελεστή εξασθένησης $a_1 = 0.6$ και $a_2 = 0.5$ κόστος 100 και 125 μονάδες αντίστοιχα. Υποθέτουμε επίσης ότι το πεδίο των αισθητήρων είναι σε ένα πλέγμα δύο διαστάσεων, με διαστάσεις $n \in \{4,5,...,40,50\}$ με κατώφλι πιθανότητας απώλειας $T = 0.01$

n	Z^*	Z_{LP}	Z_G	$n_{k=1}$	$n_{k=2}$	n_{TOT}	$t_{c\#}$(sec)	$t_{c\#}$(min)	t_M(sec)	t_M(min)
4	1100	707	1250	0	10	10	0.00	0.00	0.05	0.00
5	1450	976	1625	0	13	13	0.00	0.00	0.15	0.00
6	1875	1283	2250	0	18	18	0.00	0.00	0.42	0.01
7	2325	1628	2725	1	21	22	0.01	0.00	0.95	0.02
8	2825	2006	3125	0	25	25	0.02	0.00	1.89	0.03
9	3400	2417	3725	1	29	30	0.03	0.00	3.58	0.06
10	4000	2863	4375	0	35	35	0.05	0.00	6.38	0.11
11	4675	3343	4975	1	39	40	0.08	0.00	10.66	0.18
12	5225	3857	5625	0	45	45	0.13	0.00	17.10	0.28
13	6050	4407	6600	1	52	53	0.21	0.00	27.55	0.46
14	6775	4992	7375	0	59	59	0.34	0.01	41.76	0.70
15	7600	5612	8100	1	64	65	0.47	0.01	60.92	1.02
16	9625	6267	9225	1	73	74	0.69	0.01	89.44	1.49
17	12550	6957	9975	1	79	80	0.98	0.02	123.78	2.06
18	14725	7683	10725	1	85	86	1.33	0.02	168.97	2.82
19	15475	8443	11875	0	95	95	1.85	0.03	230.05	3.83
20	16350	9239	13225	1	105	106	2.12	0.04	316.10	5.27
21	18650	10070	14225	1	113	114	2.78	0.05	411.99	6.87
22	20675	10936	15500	0	124	124	3.73	0.06	539.66	8.99
23	26400	11837	16975	1	135	136	4.95	0.08	703.72	11.73
24	27325	12773	18225	1	145	146	6.32	0.11	896.18	14.94
25	29250	13745	19100	1	152	153	7.79	0.13	1102.10	18.37
26	31926	14751	20725	1	165	166	9.97	0.17	1400.43	23.34
27	34325	15793	22100	1	176	177	12.33	0.21	1740.54	29.01
28	43850	16870	23475	1	187	188	15.54	0.26	2153.21	35.89
29	44525	17982	24875	0	199	199	18.76	0.31	2602.60	43.38
30	46175	19129	26350	1	210	211	22.83	0.38	3220.88	53.68

Πίνακας 2 – Αποτελέσματα Αβέβαιης Κάλυψης



Σχήμα 8 – Κόστος Κάλυψης Αβέβαιης Κάλυψης



Σχήμα 9 – Χρόνος Εκτέλεσης Αβέβαιης Κάλυψης

Στην αβέβαια κάλυψη το κόστος αυξάνεται σε σχέση με την τέλεια κάλυψη, μιας και οι συνθήκες είναι περισσότερο ρεαλιστικές. Έτσι για N=30 στην αβέβαιη κάλυψη το κόστος κάλυψης είναι 26350 με χρήση 211 αισθητήρων, έναντι 192 αισθητήρων και κόστος 27350 για βέβαιη κάλυψη.

Σημαντική διαφορά υπάρχει επίσης στον αριθμό των αισθητήρων που χρησιμοποιούνται για την κάλυψη του πεδίου. Στην αβέβαιη κάλυψη επικρατεί η χρήση των αισθητήρων μεγαλύτερου κόστους (κ=2), σε αντίθεση με τους αισθητήρες μικρού κόστους που η χρήση τους παραμένει συμπληρωματική για την περίπτωση κάλυψης του τελευταίου μη καλυμμένου σημείου στο τέλος εκτέλεσης του αλγορίθμου.

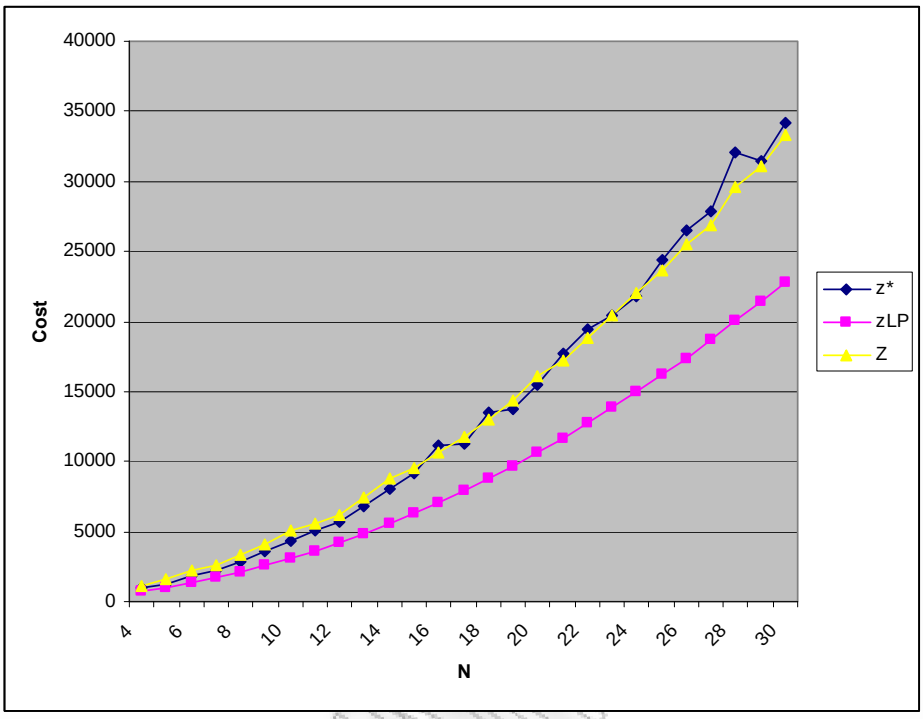
Ο χρόνος εκτέλεσης, αυξάνεται εκθετικά και στην αβέβαιη κάλυψη. Η διαφορά στον χρόνο εκτέλεσης μεταξύ των δυο διαφορετικών υλοποιήσεων εξακολουθεί να είναι μεγάλη, και υπέρ της C# όπου ο χρόνος εκτέλεσης είναι ιδιαίτερα μικρός.

4.4 Αβέβαιη κάλυψη με περιορισμένη ακτίνα κάλυψης

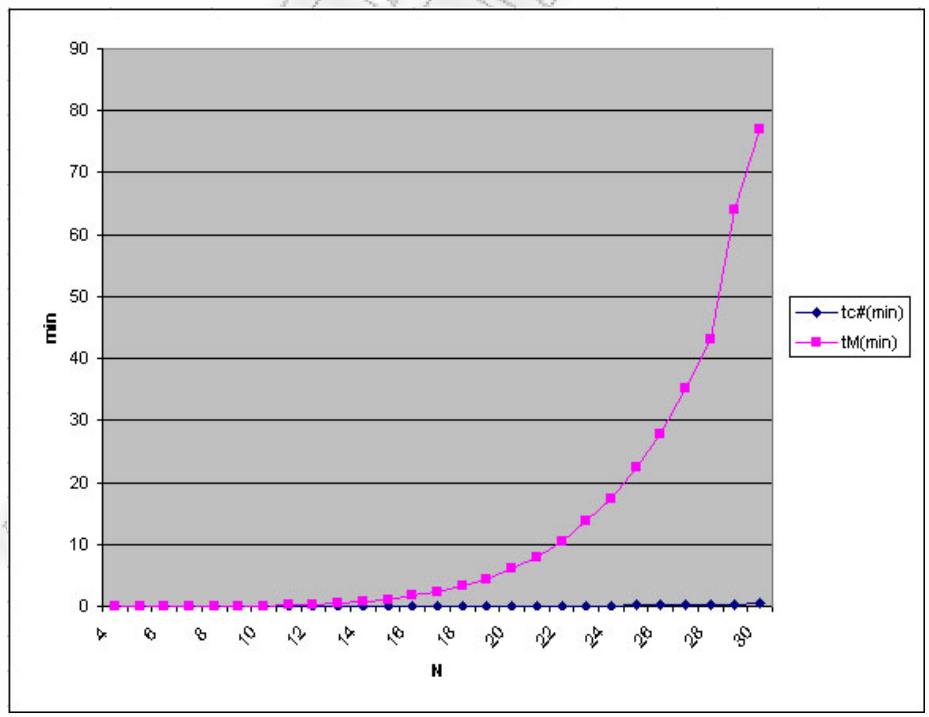
Ομοίως με το μοντέλο αβέβαιης κάλυψης, θεωρούμε 2 αισθητήρες και ακτίνα κάλυψης $r_1 = 1$ και $r_2 = 2$, συντελεστές αβεβαιότητας $r_{e1} = 0.8$, $r_{e2} = 1.6$ και ίδιο κόστος 100 και 125 μονάδες αντίστοιχα . Οι συντελεστές λ, β έχουν τιμές $\lambda_1 = 0.5, \lambda_2 = 0.6$ και $\beta_1 = 0.5, \beta_2 = 0.6$

n	z*	z_{LP}	Z_G	$n_{k=1}$	$n_{k=2}$	n_{TOT}	$t_{c\#}$(sec)	$t_{c\#}$(min)	t_M(sec)	t_M(min)
4	1000	690	1125	0	9	9	0.001	0.000	0.050	0.001
5	1250	993	1650	4	10	14	0.001	0.000	0.158	0.003
6	1825	1327	2200	2	16	18	0.002	0.000	0.413	0.007
7	2250	1696	2575	2	19	21	0.006	0.000	0.905	0.015
8	2875	2108	3325	2	25	27	0.011	0.000	1.949	0.032
9	3600	2574	4075	2	31	33	0.021	0.000	3.803	0.063
10	4325	3085	5025	4	37	41	0.041	0.001	7.099	0.118
11	5075	3638	5600	1	44	45	0.066	0.001	11.633	0.194
12	5750	4236	6225	1	49	50	0.104	0.002	18.500	0.308
13	6850	4882	7400	4	56	60	0.169	0.003	30.196	0.503
14	8050	5572	8750	5	66	71	0.271	0.005	47.687	0.795
15	9175	6308	9575	2	75	77	0.393	0.007	68.727	1.145
16	11150	7089	10650	4	82	86	0.583	0.010	99.468	1.658
17	11325	7915	11800	3	92	95	0.833	0.014	140.047	2.334
18	13475	8787	12975	1	103	104	1.155	0.019	193.446	3.224
19	13775	9704	14350	6	110	116	1.61	0.027	267.445	4.457
20	15450	10667	16150	9	122	131	2.244	0.037	370.091	6.168
21	17700	11675	17175	3	135	138	2.916	0.049	477.568	7.959
22	19400	12728	18850	6	146	152	3.922	0.065	634.244	10.571
23	20475	13827	20450	7	158	165	5.126	0.085	832.823	13.880
24	21825	14971	22025	9	169	178	6.475	0.108	1037.000	17.283
25	24400	16161	23700	7	184	191	8.424	0.140	1342.000	22.367
26	26550	17396	25500	5	200	205	10.677	0.178	1674.000	27.900
27	27825	18676	26900	9	208	217	13.174	0.220	2106.000	35.100
28	32125	20002	29600	11	228	239	16.894	0.282	2583.000	43.050
29	31425	21373	31100	11	240	251	20.54	0.342	3833.000	63.883
30	34150	22789	33300	13	256	269	25.362	0.423	4623.000	77.050

Πίνακας 3 – Αποτελέσματα Αβέβαιης Κάλυψης με Περιορισμένο Εύρος



Σχήμα 10 – Κόστος Τοποθέτησης Αβέβαιης Κάλυψης με Περιορισμένο Εύρος



Σχήμα 11 – Χρόνος Εκτέλεσης Αβέβαιης Κάλυψης με Περιορισμένο Εύρος

Διαφοροποιημένο ως προς την ακτίνα κάλυψης το μοντέλο αυτό, προσεγγίζει περισσότερο ρεαλιστικά το πρόβλημα κάλυψης. Έτσι και το κόστος, είναι μεγαλύτερο από τα προηγούμενα μοντέλα, αν και η διαφορά με το μοντέλο αβέβαιης κάλυψης είναι ελάχιστη. Έτσι για n=30 κόστος είναι 33300.

Και για την αβέβαιη κάλυψη με περιορισμένη ακτίνα ο χρόνος εκτέλεσης σε c# είναι πολύ καλύτερος σε σχέση με αυτόν από της υλοποίησης σε Matlab. Ο χρόνος αυξάνεται εκθετικά σε σχέση με το μέγεθος του grid, φτάνοντας στα 77 λεπτά για το Matlab και στα 0,42 λεπτά για την C# για n=30.

Σε αντίθεση με τον αλγόριθμο αβέβαιης κάλυψης, η χρήση των αισθητήρων μικρότερης χρέωσης είναι σαφώς μεγαλύτερη, σχεδόν για κάθε μέγεθος του grid φτάνοντας έως και τους 13 αισθητήρες για n=30.

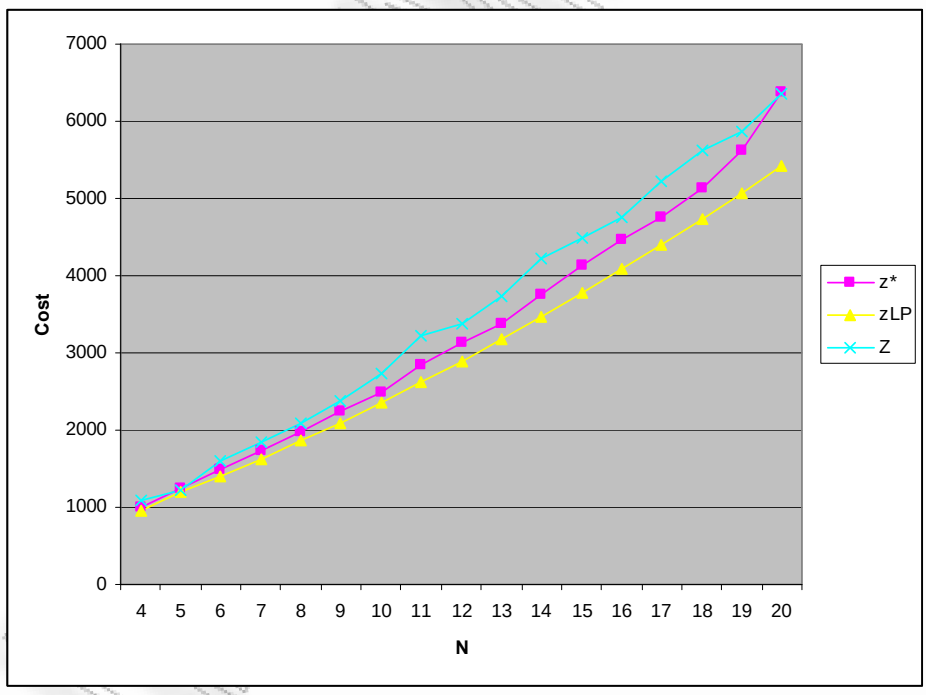
4.5 Log-Normal Shadowing

Η διαφοροποίηση στον αλγόριθμο αυτό είναι ότι οι συνθήκες είναι πλέον ρεαλιστικές, κατά συνέπεια και τα χαρακτηριστικά των αισθητήρων προς τοποθέτηση. Για λόγους σύγκρισης, κρατάμε το κόστος των αισθητήρων στις τιμές των δυο προηγούμενων περιπτώσεων κάλυψης, δηλαδή $c_1 = 100$ και $c_2 = 150$ μονάδες. Θεωρούμε ότι ο συντελεστής απωλειών είναι 2.0, και ότι οι συχνότητες λειτουργίας των αισθητήρων είναι $f_1 = 900Mhz$ και $f_2 = 413MHz$. Η μέση τιμή της λογαριθμικής κατανομής είναι μηδενική, και η τυπική απόκλιση ίση με $\sigma = 5dB$.

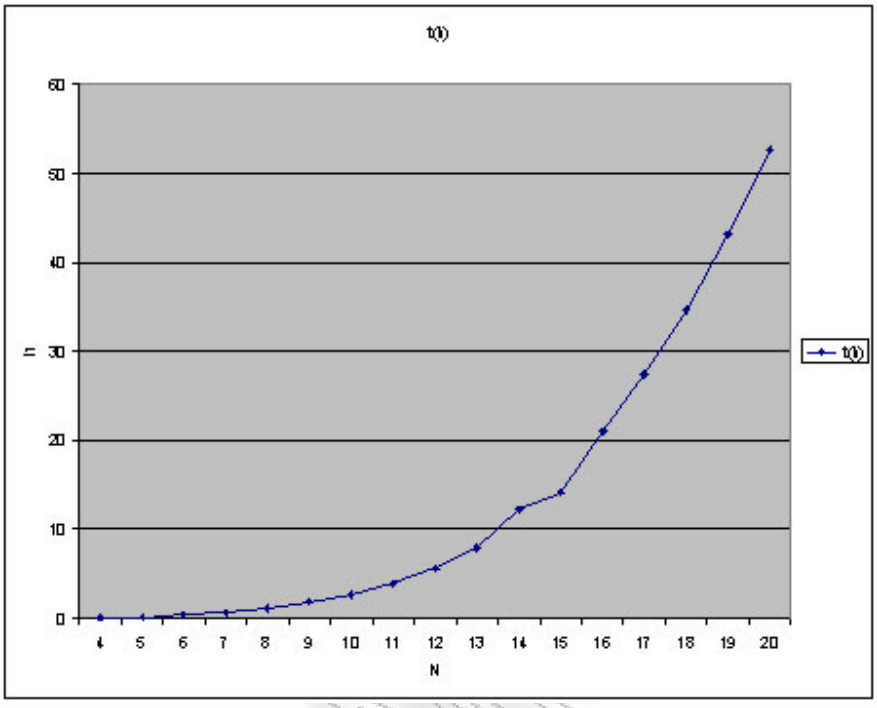
Η κρίσιμη απόσταση συσχέτισης d^* είναι $d_1^* = 10m$ για τον πρώτο αισθητήρα, και $d_2^* = 20m$ για τον δεύτερο. Η απόσταση μεταξύ δυο γειτονικών κόμβων του πλέγματος, θεωρούμε ότι είναι ίση με την απόσταση συσχέτισης $d^* = 10m$ [14] του αισθητήρα με το μικρότερο κέρδος.

n	z*	z_{LP}	Z	nk=1	nk=2	nTOT	t(sec)	t(min)	t(h)
4	1000	952	1100	1	8	9	218.31	3.64	0.06
5	1250	1189	1225	1	9	10	551.27	9.19	0.15
6	1500	1389	1600	1	12	13	1181.48	19.69	0.33
7	1725	1616	1850	1	14	15	2253.56	37.56	0.63
8	1975	1856	2100	1	16	17	3911.00	65.18	1.09
9	2250	2099	2375	0	19	19	6373.00	106.22	1.77
10	2500	2355	2725	1	21	22	9704.49	161.74	2.70
11	2850	2618	3225	1	25	26	14189.52	236.49	3.94
12	3125	2892	3375	0	27	27	20317.16	338.62	5.64
13	3375	3177	3725	1	29	30	28671.30	477.85	7.96
14	3750	3470	4225	1	33	34	43779.04	729.65	12.16
15	4125	3908	4500	0	36	36	50869.46	847.82	14.13
16	4475	4082	4750	0	38	38	75390.00	1256.50	20.94
17	4750	4401	5225	1	41	42	98607.83	1643.46	27.39
18	5125	4731	5625	0	45	45	124606.26	2076.77	34.61
19	5625	5069	5875	0	47	47	155543.05	2592.38	43.21
20	6375	5417	6350	0	50	50	189512.63	3158.54	52.64

Πίνακας 4 – Αποτελέσματα Κάλυψης με Σκίαση



Σχήμα 12 - Κόστος Τοποθέτησης (LogNormal)



Σχήμα 13 - Χρόνος Εκτέλεσης (LogNormal)

Ο υπολογισμός του κόστους λαμβάνοντας υπόψη την χωρική συσχέτιση, απαιτεί την μεγαλύτερη υπολογιστική ισχύ και χρόνο. Είναι χαρακτηριστικό ότι για το $20x20$ ο χρόνος που απαιτείται είναι 52,64 ώρες. Και στο μοντέλο αυτό ο χρόνος αυξάνεται εκθετικά (Σχήμα 12).

Όπως και στο μοντέλο αβέβαιης κάλυψης, η χρήση των αισθητήρων κόστους 100 μονάδων (κ=1) είναι σχεδόν ανύπαρκτη. Η χρήση τους παραμένει μόνο για κάλυψη του τελευταίου ακάλυπτου σημείου πριν το τερματισμό του αλγορίθμου.

5. Συμπεράσματα

5.1 Ανασκόπηση

Σε αυτήν την εργασία εξετάσαμε το πρόβλημα κάλυψης ενός πεδίου, το οποίο αντιπροσωπεύεται από ένα πλέγμα, με χρήση περισσότερων του ενός τύπων αισθητήρων. Για τον υπολογισμό του ελάχιστου αριθμού αισθητήρων και κόστους, αναπτύχθηκαν βασικές στρατηγικές τοποθέτησης για τέλεια και αβέβαιη κάλυψη. Για την καλύτερη προσέγγιση του προβλήματος κάλυψης, το μοντέλο της αβέβαιης κάλυψης επεκτάθηκε σε αβέβαιη κάλυψη με περιορισμένη ακτίνα κάλυψης, και αναπροσδιορίστηκε με την χρήση μικρής κλίμακας απωλειών για να καλύψει ρεαλιστικά το φαινόμενο των απωλειών.

Για κάθε μια από τις παραπάνω προσεγγίσεις, αναπτύχθηκε ένα κοινό πλαίσιο λύσης στηριζόμενο στον αλγόριθμο greedy. Για την επίλυση των αλγορίθμων ανά μοντέλο κάλυψης, έγινε επίλυση σε περισσότερες από μια γλώσσες προγραμματισμού, εξασφαλίζοντας έτσι την εγκυρότητα των αποτελεσμάτων και καλύτερη απόδοση.

5.2 Συμπεράσματα

Τα συμπεράσματα στα οποία καταλήγουμε από την μελέτη των τεσσάρων μοντέλων διάδοσης, συμπεραίνουμε ότι όσο πιο ρεαλιστικό είναι το μοντέλο κάλυψης, τόσο περισσότεροι αισθητήρες με μεγαλύτερο κόστος τοποθετούνται στο δίκτυο. Έτσι παρατηρούμε ότι ενώ στο μοντέλο τέλειας κάλυψης για $N = 30$ απαιτούνται 29 αισθητήρες τύπου $k = 1$ και 163 τύπου $k = 2$, για την περίπτωση της αβέβαιης κάλυψης απαιτούνται 1 και 210 αντίστοιχα. Ομοίως και για το μοντέλο σκίασης.

Λόγο της πιθανοτικής κάλυψης, και το μικρό κατώφλι πιθανότητας απώλειας ενός στόχου, είναι μεγαλύτερο το κέρδος από την τοποθέτηση αισθητήρα με μεγαλύτερο εύρος, άρα και κόστος, σε σχέση με έναν αισθητήρα φθηνότερο αλλά με μικτότερο κόστος. Έτσι παρατηρούμε ότι στους αλγόριθμους με αβέβαιη κάλυψη η χρήση των αισθητήρων μικρότερου κόστους προτιμάται μόνο όταν η χρήση τους για κάλυψη αφορά πεδίο με εύρος στα πλαίσια των δυνατοτήτων τους.

Σχετικά με τον χρόνο εκτέλεσης του αλγορίθμου, η επιλογή μιας γλώσσας τρίτης γενιάς για μεγάλου βαθμού δίκτυα κρίνεται απαραίτητη. Έτσι μπορεί να αυξηθεί ο βαθμός του πλέγματος, παραμένοντας μικρός ο χρόνος εκτέλεσης.

Η διαφοροποίηση των αποτελεσμάτων για την τέλεια και την αβέβαιη κάλυψη από αυτά που αρχικά είχαν υπολογιστεί [5], οφείλεται στο γεγονός οτι η επιλογή του σημείου για τοποθέτηση αισθητήρα, όταν υπάρχουν παραπάνω από ένα ισοδύναμα σημεία δεν έχει διευκρινιστεί. Στην υλοποίηση που έχουμε παρουσιάσει, γίνεται επιλογή πάντα του πρώτου σημείου. Αν όμως διαφοροποιηθεί η επιλογή αυτή, τα αποτελέσματα ενδέχεται να διαφέρουν.

5.3 Προτάσεις για περαιτέρω μελέτη

Η περαιτέρω μελέτη του μοντέλου κάλυψης, θα πρέπει να σχετίζεται με την επέκταση του σε περισσότερο ρεαλιστικές συνθήκες. Οι μικρής κλίμακας διαλείψεις που προκαλούνται από τις πολλαπλές διαδρομές του μέσου μετάδοσης ή αλλιώς από την πολυδιαδρομική διάδοση έχουν επιπτώσεις στην συμπεριφορά του καναλιού και κατά συνέπεια στην ποιότητα του σήματος [15]. Έτσι θα πρέπει να συνυπολογιστούν στην πιθανότητα κάλυψης ενός σημείου από έναν αισθητήρα.

Επίσης ένας ακόμα παράγοντας που θα πρέπει να ληφθεί υπόψη κατά την τοποθέτηση είναι η μεγιστοποίηση της διάρκειας ζωής των αισθητήρων. Ο χρόνος ζωής ενός αισθητήρα σε ένα δίκτυο ασύρματων αισθητήρων, εξαρτάται από το μέγεθος του δικτύου και τον αριθμό των αισθητήρων που εκτείνονται σε αυτό [16].

Έτσι κατά την τοποθέτηση, θα πρέπει να ληφθεί υπόψη εκτός από την μείωση του κόστους, και η μεγιστοποίηση του χρόνου ζωής ενός αισθητήρα.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΑ

Βιβλιογραφικές Αναφορές

[1] Ohnson Kuruvila, Amiya Nayak, and Ivan Stojmenovic, Progress based localized power and cost aware routing algorithms for ad hoc and sensor wireless networks, International Journal of Distributed Sensor Networks , Vol. 2, No. 2, April-June 2006, 147-159, 2006. Further generalization is in article [S-n] listed under 'Sensor networks'.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, “Wireless sensor networks: A survey,” Computer Networks, vol. 38, no. 4, pp. 393–422, Mar. 2002.

[3] My T. Thai, Feng Wang, Hongwei Du, and Xiaohua Jia, "Coverage Problems in Wireless Sensor Networks: Designs and Analysis", International Journal of Sensor Networks, special issue on Coverage Problems in Sensor Networks, 2007.

[4] Hossein Pishro-Nik, Kevin Chan, Faramarz Fekri, On Connectivity Properties of Large-Scale Sensor Networks, October 2004

[5] Altınel İ.K., Aras N., Güney E., Ersoy C. Effective coverage in sensor networks: Binary integer programming formulations and heuristics, Proc. of 2006 IEEE International Conference on Communications, 11-15 June 2006, İstanbul, Turkey.

[6] S. Meguerdichian, F. Koushanfar, M. Potkonjak, M.B. Srivastava, "Coverage Problems in Wireless Ad-Hoc Sensor Networks," IEEE Infocom 2001,Vol 3, pp. 1380-1387, April 2001.

[7] A. Fanimokun and J. Frolik, Effects of natural propagation environments on wireless sensor network coverage area, 2003 Southeastern Symposium on System Theory (SSST03), Morgantown, WV, March 16-18, 2003.

[8] K. Chakrabarty, S.S.Iyengar, Hairong Qi, and E.C. Cho, “Grid Coverage of Surveillance and Target Location in Distributed Sensor Networks”, IEEE Transactions on Computers, Vol 51, No. 12, December 2002.

[9] S. S. Dhillon, K. Chakrabarty and S. S. Iyengar, "Sensor placement for grid coverage under imprecise detections", Proc. International Conference on Information Fusion (FUSION 2002), pp. 1581-1587, 2002.

[10] S. S. Dhillon and K. Chakrabarty, "Sensor placement for effective coverage and surveillance in distributed sensor networks", Proc. IEEE Wireless Communications and Networking Conference, pp. 1609--1614, 2003.

[11] Y. Zou and K. Chakrabarty, "Sensor deployment and target localization. based on virtual forces" in Proceedings of INFOCOM, March 2003.

[12] Guiling Wang, Guohong Cao, and Tom La Porta, "Movement-Assisted Sensor Deployment", IEEE Transactions on Computers, Vol 5, No. 6, 2006

[13] Lifetime Evaluation and Spatial Correlation Effects on Wireless Sensor Networks, G. Bravos, A. Kanatas, A. Kalis. Proceedings of 15th IST mobile & Wireless Communications Summit, 4 – 8 June, 2006

[14] M. Gudmundson. Correlation model for shadow fading in mobile radio systems. Electronics Letters, vol.27, no.23, pp.2145-2146, 1991.

[15] M. Haenggi, "On Routing in Random Rayleigh Fading Networks", *IEEE Transactions on Wireless Communications*, vol. 4, pp. 1553-1562, July 2005

[16] Sunil Kandukuri and Stephen Boyd, "Optimal power control in interference limited fading wireless channels with outage probability specifications." IEEE Trans. Wireless Comm., VOL. 1, NO. 1, pp. 46-55, Jan. 2002.

[17] Thomas H. Cormen, Charles E. Leiserson, Roland L. Rivest and Clifford Stein, "Introduction to Algorithms". Second Edition, 2001

[18] LINGO - The Modeling Language and Optimizer. LINDO Systems Inc., Chicago, 2002.

[19] Optimization Modeling with LINGO, LINDO Systems, Inc., 5th ed., USA, 2002

[20] Ayad Salhieh, Jennifer Weinmann, Manish Kochhal, Loren Schwiebert, "Power Efficient Topologies forWireless Sensor Networks". Parallel Processing, International Conference on, 2001.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

Παράρτημα

Κώδικάς Matlab επίλυσης του Greedy αλγορίθμου για τέλεια κάλυψη

```
%Greedy Heuristic for Precise Detection
clc;
clear;
format long;

%grid dimensions
V = horzcat([4:30]);
%Greedy Results
zg = zeros(1, length(V));

%CPU Time
t = zeros(1, length(V));

%Number of Sensors types
K=2;
%Minimum Number of sensors cover a point j
b=2;
%sensors ranges
d = [1 2];
%sensors cost
c = [100 150];

%debug=1 => create text files
debug=0;

if debug==1
     %Gain Factors / Coverage / Sensors Location
     file_1 = fopen('ecp1_log1.txt','w');

     %Coverage coefficient file
     file_2 = fopen('ecp1_log2.txt','w');
end

for n = 1:length(V);
    %Grid Dimension
    N = V(1,n);

    %Grid Point - (1,1): Sensor (1,2): Sensor Type
    x = repmat({[0 0]}, N, N);

    %Coverage coefficients
    A = repmat({zeros(N,N)},N,N);

    %Coverage
```

```
covj = zeros(N,N);

%Gain Factor
gik = zeros(N,N,K);

%CPY Time start
tic

%For every type of sensor
for k=1:K;
    %for all greedy points
    for i=1:N;
        for j=1:N;

            %Temporary Array of coverage
            A1=zeros(N,N);

            %Current point coverage coefficient is 1
            A1(i,j)=1;

            %Calculate coverage coefficient if sensor is in (i,j)
            for xi=1:N;
                for yj=1:N;
                    %Euclidean distance
                    if sqrt(((i-xi)^2)+((j-yj)^2))<=d(k);
                        A1(xi,yj)=1;
                    end
                end
            end

            if debug==1
                fprintf(file_2, 'i=%d, j=%d\n', i,j);
                for w=1:N
                    fprintf(file_2, '%d ', A1(w,:));
                    fprintf(file_2, '\n');
                end
                fprintf(file_2, '\n');
            end


            %Insert into Coverage coefficient Array, temp array A1
            A(i,j,k) = {A1};
        end
    end
end

%Coverage Initialization
cov = zeros(N,N);

%While exist point with coverage less than b
while (find(cov<b))

    %Initialize Gain Factor
    gik = zeros(N,N,K);
```

UNDER THE COPYRIGHT

```
        % point
        point = ones(1,3);
        maximum = gik(1,1,1);

        %For every type of sensor
        for i=1:N;
            %For every point
            for j=1:N;
                for k=1:K;
                   %if point is available
                    if x{i,j}(1,1) ~= 1
                        for xi=1:N;
                            for yj=1:N;
                                %Calculate Gain Factor
                                gik(i,j,k) = gik(i,j,k) +
A{xi,yj,k}(i,j) * max(0, b - cov(xi,yj))/c(k);
                            end
                        end
                    end

                    if (gik(i,j,k)>maximum)
                       point = [i,j,k];
                       maximum = gik(i,j,k);
                    end

                end
            end
        end


        %Place Sensor
        x(point(1), point(2)) = {[1,point(3)]};

        %Update points remaining coverage
        cov = zeros(N,N);

        %Calculate Coverage
        for i=1:N;
            for j=1:N;
                if x{i,j}(1,1)==1
                    cov = cov + A{i,j,x{i,j}(1,2)} ;
                end
            end
        end

        if debug==1
            for i=1:N;
                for j=1:N;
                    X(i,j) = x{i,j}(1,2);
                end
            end

            for w=1:N
                fprintf(file_1, '%1.2f ', gik(w,:,1));
```

UNDER THE OPEN

```
                fprintf(file_1, '  ');
                fprintf(file_1, '%1.2f ', gik(w,:,2));
                fprintf(file_1, '  ');
                fprintf(file_1, '%d ', X(w,:));
                fprintf(file_1, '            ');
                fprintf(file_1, '%d ', cov(w,:));
                fprintf(file_1, '\n');
            end
            fprintf(file_1, '\n');
       end
    end

    %Close Files Opened
    if debug==1
        fclose(file_1);
        fclose(file_2);
    end

    %CPU Time End
    t(1,n) = toc;

    %Display Results
    for i=1:N;
        for j=1:N;
            X(i,j) = x{i,j}(1,1);
            S(i,j) = x{i,j}(1,2);
            if x{i,j}(1,1)==1;
                zg(1,n) = zg(1,n) + c(1,x{i,j}(1,2));
            end
        end
    end

    zg
    t

end
```

Κώδικάς Matlab επίλυσης γραμμικού προβλήματος για τέλεια κάλυψη

```
%Solve the linear programming problem of ECP1
clear;
clc;

%debug - Lingo
debug = 1;

%sensors ranges
d=[1 2];

%Number of Sensors types
K=2;
%Minimum Number of sensors cover a point j
b=2;

%sensors cost
c=[100 150];

%grid dimensions
V = horzcat([4:30]);
%Objective Values
Fval = zeros(1,length(V));

%For sll grid dimensions
for n = 1:length(V)
    %Grid Dimension
    N = V(1,n);

    %Initialize Coverage Coefficient Matrice
    A = zeros(1,N*N*2);

    %Matrice Used for Constraint (3)
    G = zeros(1,N*N*2);

    %Objective function
    f = [];
    for k=1:K
        f = horzcat(f,repmat(c(1,k),1,N*N));
    end

    if debug==1
        file_1 = fopen('lingo_ecp1.lg4','w');
        fprintf(file_1, 'MIN=');

        for i=1:2*N*N
            fprintf(file_1, '%5.4f*x%d', f(1,i), i);
            if i<2*N*N
                fprintf(file_1, '+');
            end
        end
```

```
    fprintf(file_1, ';\n');
end

%constraint (2)
%For all types of sensors
for k=1:K;
    %Helper
    z=0;
    %For all greedy points
    for i=1:N;
        for j=1:N;

            %Coverage If Sensor is Places at point (i,j)
            A1=zeros(N,N);
            A1(i,j)=1;

            %Sensor Location
            A2=zeros(N,N);
            A2(i,j)=1;

            %Calculate coverage coefficient if sensor is in (i,j)
            for x=1:N;
                for y=1:N;
                    %Euclidean distance
                    if sqrt(((i-x)^2)+((j-y)^2))<=d(k);
                        A1(x,y)=1;
                    end
                end
            end

            z=z+1;

            %Reshape Matrices for linprog
            A(z,N*N*(k-1)+1:N*N*k) = reshape(A1,1,N*N);
            G(z,N*N*(k-1)+1:N*N*k) = reshape(A2,1,N*N);

        end
    end
end

%Left part of inequality
A = [-A ; G];

%Right part of inequality
B = ones(1,K*N*N);
B(1,1:N*N) = -1 * b;

%Create lingo file
if debug==1
    TX = [-A ; G];

    for i=1:2*N*N;
        for j=1:2*N*N
```

```
                fprintf(file_1, '%5.4f*x%d', TX(i,j),j);
                if j<2*N*N
                    fprintf(file_1, '+');
                end
            end

            if i>N*N
                fprintf(file_1, '<=%5.4f;', B(1,i));
            else
                fprintf(file_1, '>=%5.4f;', -B(1,i));
            end

            fprintf(file_1, '\n') ;
        end

        for i=1:2*N*N
            %fprintf(file_1, '@BIN(x%d);', i);
            %fprintf(file_1, '\n') ;
        end

        fclose(file_1);
    end

    %Lower and Upper bounds
    lb=zeros(K*N*N,1);
    ub=ones(K*N*N,1);

    %Solve the linear programming problem
    [x, fval, exitflag, output]=linprog(f, A, B,[],[], lb, ub);

    %Update Results Metrice
    Fval(1,n) = fval;
end

%Results
Fval
```

UNEDITED COPY

Κώδικάς Matlab επίλυσης του Greedy αλγορίθμου για αβέβαιη κάλυψη

```
format long;
%Greedy Heuristic for Uncertain Detection
clc;
clear;

%grid dimensions
V = horzcat([4:30]);
%Greedy Results
zg = zeros(1, length(V));

%CPU Time
t = zeros(1, length(V));

%Number of Sensors types
K=2;

%miss probability threshold
T=0.01;

%expnentil decay parameters
a = [0.6 0.5];

%sensors cost
c = [100 125];

%debug=1 => create text files
debug=0;

if debug==1
     %Gain Factors / Coverage / Sensors Location
     file_1 = fopen('ecp2_log1.txt','w');

     %Coverage coefficient file
     file_2 = fopen('ecp2_log2.txt','w');
end

for n = 1:length(V);
    %Grid Dimension
    N = V(1,n);

    %Grid Point - (1,1): Sensor (1,2): Sensor Type
    x = repmat({[0 0]}, N, N);

    %Coverage
    covj = zeros(N,N);

    %Gain Factor
    gik = zeros(N,N,K);
```

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΑ

```
    %Probability of sensing point j with a sensor of type k located at
point i
    P = repmat({zeros(N,N)},N,N);

    tic
    %For every type of sensor
    for k=1:K;
        for i=1:N;
            for j=1:N;

                %Temporary Array - Propability
                P1=zeros(N,N);
                Dist=zeros(N,N);
                %Calculate probability if sensor is in (i,j)
                for xi=1:N;
                    for yj=1:N;

                        %Euclidean distance
                        eud = sqrt(((i-xi)^2)+((j-yj)^2));
                        Dist(xi,yj) = eud;
                            if (eud>0)
                                P1(xi,yj) = exp(-a(1,k)*eud);
                            else
                                P1(xi,yj) = 0.99;
                            end
                    end
                end

                if debug==1
                    fprintf(file_2, 'i=%d, j=%d\n', i,j);
                    for w=1:N
                        fprintf(file_2, '%d ', P1(w,:));
                        fprintf(file_2, '\n');
                    end
                    fprintf(file_2, '\n');
                end

                %Insert into Probability Array temp array P1
                P(i,j,k) = {P1};
            end
        end
    end

    %Miss Probability
    missj = ones(N,N);

    %While Miss Probability is less than threshold
    while (find(missj>T))

        %Initialize Gain Factor
        gik = zeros(N,N,K);
        % Initialize Miss Probability
        missj = ones(N,N);

        for k=1:K;
```

```
            for i=1:N;
                for j=1:N;
                    if x{i,j}(1,2)==k
                        missj = missj .* (1 - P{i,j,k});
                    end
                end
            end
        end

        % point
        point = ones(1,3);
        maximum = gik(1,1,1);

        %Gain Factor
        %For every type of sensor
        for i=1:N;
            %For every point
            for j=1:N;
                for k=1:K;
                   %if point is available
                    if x{i,j}(1,1) ~= 1
                        for xi=1:N;
                            for yj=1:N;
                                %Calculate Gain Factor
                                gik(i,j,k) = gik(i,j,k) +
P{xi,yj,k}(i,j)*(max(0, missj(xi,yj)- T));
                            end
                        end
                    end

                    if (gik(i,j,k)>maximum)
                       point = [i,j,k];
                       maximum = gik(i,j,k);
                    end

                end
            end
        end


        %Place Sensor
        x(point(1), point(2)) = {[1,point(3)]};

        missj = ones(N,N);

        %Update points remaining coverage
        for k=1:K;
            for i=1:N;
                for j=1:N;
                    if x{i,j}(1,2)==k
                        missj = missj .* (1 - P{i,j,k});
                    end
                end
            end
        end
```

UNDER CONSTRUCTION

```
    end

    %Close Files Opened
    if debug==1
        fclose(file_1);
        fclose(file_2);
    end

    t(1,n) = toc;

    %Display Results
    for i=1:N;
        for j=1:N;
            X(i,j) = x{i,j}(1,1);
            S(i,j) = x{i,j}(1,2);
            if x{i,j}(1,1)==1;
                zg(1,n) = zg(1,n) + c(1,x{i,j}(1,2));
            end
        end
    end

    zg
    t

end
```

DANE TWOJEGO TEPAK

Κώδικάς Matlab επίλυσης γραμμικού προβλήματος για αβέβαιη κάλυψη

```
format long;
%Solve the linear programming problem of ECP2
clear;
clc;

%Debug - lingo
debug=1;

%Number of Sensors types
K=2;

%miss probability threshold
T=0.01;

%sensors cost
c=[100 125];

%expnentil decay parameters
a=[0.6 0.5];

%grid dimensions
V = horzcat([19]);
%Objective Values
Fval = zeros(1,length(V));

%For sll grid dimensions
for n = 1:length(V)
    %Grid Dimension
    N = V(1,n);

    %Initialize Coverage Coefficient Matrice
    A = zeros(1,N*N*K);

    %Matrice Used for Constraint (3)
    G = zeros(1,N*N*K);

    %Objective function
    f = [];
    for k=1:K
        f = horzcat(f,repmat(c(1,k),1,N*N));
    end

    if debug==1
        file_1 = fopen('lingo_ecp2_19.lg4','w');
        fprintf(file_1, 'MIN=');

        for i=1:2*N*N
            fprintf(file_1, '%5.4f*x%d', f(1,i), i);
```

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

```
            if i<2*N*N
                fprintf(file_1, '+');
            end
        end

        fprintf(file_1, ';\n');
    end

    %constraint (2)
    %For all types of sensors
    for k=1:K;
        %Helper
        z=0;
        %For all greedy points
        for i=1:N;
            for j=1:N;

                %Coverage If Sensor is Places at point (i,j)
                A1=zeros(N,N);
                A2=zeros(N,N);

                %Sensor Location
                A2(i,j)=1;

                %Calculate coverage coefficient if sensor is in (i,j)
                for x=1:N;
                    for y=1:N;
                        %Euclidean distance
                        eud = sqrt(((i-x)^2)+((j-y)^2));

                        %Probability of sensing point j with a sensor
of type k located at point i
                        if (eud>0)
                            pijk = exp(-a(1,k)*eud);
                        else
                            pijk = 0.99;
                        end

                        A1(x,y) = -log(1-pijk);

                    end
                end

                z=z+1;

                %Reshape Matrices for linprog
                A(z,N*N*(k-1)+1:N*N*k) = reshape(A1,1,N^2);
                G(z,N*N*(k-1)+1:N*N*k) = reshape(A2,1,N^2);

            end
        end
    end

    %Right part of inequality
```

UNEDITIONED DRAFT

```
    B = ones(1,K*N^2);
    B(1,1:N*N) = log(T);

    %Create lingo file
    if debug==1
        TX = [-A ; G];

        for i=1:2*N*N;
            for j=1:2*N*N;
                fprintf(file_1, '%5.4f*x%d', TX(i,j),j);
                if j<2*N*N
                    fprintf(file_1, '+');
                end
            end

            if i>N*N
                fprintf(file_1, '<=%5.4f;', B(1,i));
            else
                fprintf(file_1, '<=%5.4f;', B(1,i));
            end

            fprintf(file_1, '\n') ;
        end

        for i=1:2*N*N
            fprintf(file_1, '@BIN(x%d);', i);
            fprintf(file_1, '\n') ;
        end

        fclose(file_1);
    end

    %Right part of inequality
    A = [-A ; G];

    %Lower and Upper bounds
    lb=zeros(K*N^2,1);
    ub=ones(K*N^2,1);

    %Solve the linear programming problem
    [x, fval, exitflag, output]=linprog(f, A, B,[],[], lb, ub);

    %Update Results Metrice
    Fval(1,n) = fval;
end

%Results
Fval
```

UNIVERSITY OF PERAK

Κώδικάς Matlab επίλυσης γραμμικού προβλήματος για κάλυψη με περιορισμένη ακτίνα

```
format long;
%Greedy Heuristic for Uncertain Detection
clc;
clear;

%grid dimensions
V = horzcat([4:30]);
%Greedy Results
zg = zeros(1, length(V));

%CPU Time
t = zeros(1, length(V));

%Number of Sensors types
K=2;

%miss probability threshold
T=0.01;

%l, b
l=[0.5, 0.6];
b=[0.5, 0.6];

%r ,re
r=[1 2];
re=[0.8 1.6];

%sensors cost
c = [100 125];

%debug=1 => create text files
debug=0;

if debug==1
     %Gain Factors / Coverage / Sensors Location
     file_1 = fopen('ecp2_log1.txt','w');

     %Coverage coefficient file
     file_2 = fopen('ecp2_log2.txt','w');
end

for n = 1:length(V);
    %Grid Dimension
    N = V(1,n);

    %Grid Point - (1,1): Sensor (1,2): Sensor Type
    x = repmat({[0 0]}, N, N);
```

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

```
    %Coverage
    covj = zeros(N,N);

    %Gain Factor
    gik = zeros(N,N,K);

    %Probability of sensing point j with a sensor of type k located at
point i
    P = repmat({zeros(N,N)},N,N);

    tic
    %For every type of sensor
    for k=1:K;
        for i=1:N;
            for j=1:N;

                %Temporary Array - Propability
                P1=zeros(N,N);

                %Calculate probability if sensor is in (i,j)
                for xi=1:N;
                    for yj=1:N;

                        eud = sqrt(((i-xi)^2)+((j-yj)^2));

                        rre_add = r(1,k) + re(1,k);
                        rre_diff = r(1,k) - re(1,k);

                        %Probability of sensing point j with a sensor
of type k located at point i
                        if (rre_add<=eud)
                            P1(xi,yj) = 0;
                        elseif (rre_diff<eud & eud<rre_add)
                            a = eud - rre_diff;
                            P1(xi,yj) = exp(-l(1,k)*(a^b(1,k)));
                        elseif (rre_diff>=eud)
                            P1(xi,yj) = 0.99;
                        end

                    end
                end

                if debug==1
                    fprintf(file_2, 'i=%d, j=%d\n', i,j);
                    for w=1:N
                        fprintf(file_2, '%d ', P1(w,:));
                        fprintf(file_2, '\n');
                    end
                    fprintf(file_2, '\n');
                end

                %Insert into Probability Array temp array P1
                P(i,j,k) = {P1};
            end
        end
```

UNDER CONSTRUCTION

```
    end

    %Miss Probability
    missj = ones(N,N);

    %While Miss Probability is less than threshold
    while (find(missj>T))

        %Initialize Gain Factor
        gik = zeros(N,N,K);
        % Initialize Miss Probability
        missj = ones(N,N);

        for k=1:K;
            for i=1:N;
                for j=1:N;
                    if x{i,j}(1,2)==k
                        missj = missj .* (1 - P{i,j,k});
                    end
                end
            end
        end

        %Gain Factor
        %For every type of sensor
        for i=1:N;
            %For every point
            for j=1:N;
                for k=1:K;
                   %if point is available
                    if x{i,j}(1,1) ~= 1
                        for xi=1:N;
                            for yj=1:N;
                                %Calculate Gain Factor
                                gik(i,j,k) = gik(i,j,k) +
P{xi,yj,k}(i,j)*(max(0, missj(xi,yj)- T));
                            end
                        end
                    end
                end
            end
        end

        %Get Maximum Gain Factors from all available sensor types
        for k=1:K;
            gm(1,k) = max(max(gik(:,:,k)));
        end


       %Get Max Gain Factor
       [i,st] = find (gm==max(max(gm)));

       %Get Sensor Location
       [i,j] = find(gik(:,:,:)==max(max(gm)));
```

UNDER CONSTRUCTION

```
        %co-ordinates
        j= j(1);
        i = i(1);
        st = st(1);

         %fix
         if (j>N);
             j = j - N;
         end

         %Place Sensor
         x(i, j) = {[1,st(1)]};

         missj = ones(N,N);

         %Update points remaining coverage
         for k=1:K;
             for i=1:N;
                 for j=1:N;
                     if x{i,j}(1,2)==k
                         missj = missj .* (1 - P{i,j,k});
                     end
                 end
             end
         end
    end

    %Close Files Opened
    if debug==1
        fclose(file_1);
        fclose(file_2);
    end

    t(1,n) = toc;

    %Display Results
    for i=1:N;
        for j=1:N;
            X(i,j) = x{i,j}(1,1);
            S(i,j) = x{i,j}(1,2);
            if x{i,j}(1,1)==1;
                zg(1,n) = zg(1,n) + c(1,x{i,j}(1,2));
            end
        end
    end

    zg
    t

end
```

UNDER THE PEAK

Κώδικάς Matlab επίλυσης γραμμικού προβλήματος για αβέβαιη κάλυψη με περιορισμένη ακτίνα

```
format long;
%Solve the linear programming problem of ECP2
clear;
clc;

%Debug - lingo
debug=1;

%Number of Sensors types
K=2;

%miss probability threshold
T=0.01;

%sensors cost
c=[100 125];

%l, b
l=[0.5, 0.6];
b=[0.5, 0.6];

%r ,re
r=[1 2];
re=[0.8 1.6];

%grid dimensions
V = horzcat([4:30]);
%Objective Values
Fval = zeros(1,length(V));

%For sll grid dimensions
for n = 1:length(V)
    %Grid Dimension
    N = V(1,n);

    %Initialize Coverage Coefficient Matrice
    A = zeros(1,N*N*K);

    %Matrice Used for Constraint (3)
    G = zeros(1,N*N*K);

    %Objective function
    f = [];
    for k=1:K
        f = horzcat(f,repmat(c(1,k),1,N*N));
    end
```

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

```
    if debug==1
        file_1 = fopen('ecp3_ip_.lg4','w');
        fprintf(file_1, 'MIN=');

        for i=1:2*N*N
            fprintf(file_1, '%5.4f*x%d', f(1,i), i);
            if i<2*N*N
                fprintf(file_1, '+');
            end
        end

        fprintf(file_1, ';\n');
    end

    %constraint (2)
    %For all types of sensors
    for k=1:K;
        %Helper
        z=0;
        %For all greedy points
        for i=1:N;
            for j=1:N;

                %Coverage If Sensor is Places at point (i,j)
                A1=zeros(N,N);
                A2=zeros(N,N);

                %Sensor Location
                A2(i,j)=1;

                %Calculate coverage coefficient if sensor is in (i,j)
                for x=1:N;
                    for y=1:N;
                        %Euclidean distance
                        eud = sqrt(((i-x)^2)+((j-y)^2));

                        rre_add = r(1,k) + re(1,k);
                        rre_diff = r(1,k) - re(1,k);

                        %Probability of sensing point j with a sensor
of type k located at point i
                        if (rre_add<=eud)
                            pijk = 0;
                        elseif (rre_diff<eud & eud<rre_add)
                            a = eud - rre_diff;
                            pijk = exp(-l(1,k)*(a^b(1,k)));
                        elseif (rre_diff>=eud)
                            pijk = 0.99;
                        end

                        A1(x,y) = -log(1-pijk);

                    end
                end
```

```
                z=z+1;

                %Reshape Matrices for linprog
                A(z,N*N*(k-1)+1:N*N*k) = reshape(A1,1,N^2);
                G(z,N*N*(k-1)+1:N*N*k) = reshape(A2,1,N^2);

            end
        end
    end

    %Right part of inequality
    B = ones(1,K*N^2);
    B(1,1:N*N) = log(T);

    %Create lingo file
    if debug==1
        TX = [-A ; G];

        for i=1:2*N*N;
            for j=1:2*N*N;
                fprintf(file_1, '%5.4f*x%d', TX(i,j),j);
                if j<2*N*N
                    fprintf(file_1, '+');
                end
            end
            if i>N*N
                fprintf(file_1, '<=%5.4f;', B(1,i));
            else
                fprintf(file_1, '<=%5.4f;', B(1,i));
            end

            fprintf(file_1, '\n') ;
        end

        for i=1:2*N*N
            fprintf(file_1, '@BIN(x%d);', i);
            fprintf(file_1, '\n') ;
        end
        fclose(file_1);
    end

    %Right part of inequality
    A = [-A ; G];

    %Lower and Upper bounds
    lb=zeros(K*N^2,1);
    ub=ones(K*N^2,1);

    %Solve the linear programming problem
    [x, fval, exitflag, output]=linprog(f, A, B,[],[], lb, ub);

    %Update Results Metrice
    Fval(1,n) = fval;
end
```

UNDER LINUX ENVIRONMENT

Κώδικάς Matlab επίλυσης γραμμικού προβλήματος για log-normal σκίαση

```
format long;
%Greedy Heuristic for Uncertain Detection
clc;
clear;

%grid dimensions
V = horzcat([4,16],20);
V=[4];
%Greedy Results
zg = zeros(1, length(V));

%CPU Time
t = zeros(1, length(V));

%Number of Sensors types
K=2;

%miss probability threshold
T=0.01;

xl=10;
Pt = [10,10];
lamda = [0.33, 0.69]; %m
std_db = 5;
b=-15;
s1=1.778;
s2=1.778;
do = [10, 20];
n=2.0;

%sensors cost
c = [100 125];

%debug=1 => create text files
debug=0;

if debug==1
     %Gain Factors / Coverage / Sensors Location
     file_1 = fopen('ecp2_log1.txt','w');

     %Coverage coefficient file
     file_2 = fopen('ecp2_log2.txt','w');
end

for n = 1:length(V);
    %Grid Dimension
    N = V(1,n);

    %Grid Point - (1,1): Sensor (1,2): Sensor Type
    x = repmat({[0 0]}, N, N);
```

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

```
    %Coverage
    covj = zeros(N,N);

    %Gain Factor
    gik = zeros(N,N,K);

    %Probability of sensing point j with a sensor of type k located at
point i
    P = repmat({zeros(N,N)},N,N);

    tic
    %For every type of sensor
    for k=1:K;
        for i=1:N;
            for j=1:N;

                Dist = zeros(N,N);
                p2p = zeros(1, 1);
                p=1;

                %Temporary Array - Probability
                P1=zeros(N,N);

                %Calculate probability if sensor is in (i,j)
                for xi=1:N;
                    for yj=1:N;
                        eud = sqrt(((i*xl-xi*xl)^2)+((j*xl-yj*xl)^2));
                        Dist(xi,yj)=eud;

                        if (find(p2p==eud))
                            %
                        else
                            p2p(1, p) = eud;
                            p=p+1;
                        end
                    end
                end


                P1(i, j) = 0.99;

                for pd=1:p-1
                    [cx,cy] = find(Dist==p2p(pd));

                    if length(cx)==1 && Dist(cx(1),cy(1))~=0 &&
p2p(pd)~=0
                        ceud = 0;
                        dij=[ceud];

                        dti=p2p(pd);
                        dtj=p2p(pd);
                        lognorm1=log10(lognrnd(0,5));
```

```
                        p=exp(-dij./do(k));

                        for cj=1:10000
                            for ci=1:length(dij)
                                norm1=normrnd(0,1);
                                norm2=normrnd(0,1);


X1correlatedt(ci,cj)=s1*(sqrt(abs(p(ci)))*norm1+sqrt(1-
p(ci)*p(ci))*norm2);
                            end
                        end


PLc1t=10*n*log10(4*pi*dti/lamda(k))+X1correlatedt;

                        mPLc1=sum(PLc1t')./10000;

                        qx = (b-(Pt(k) - mPLc1))/std_db;
                        P1(cx(1), cy(1)) = qfunc(qx);
                    end

                    for (cp=2:length(cx))
                        ceud = sqrt(((cx(cp)-cx(cp-1))^2)+((cy(cp)-
cy(cp-1))^2));
                        dij=[ceud];

                        dti=p2p(pd);
                        dtj=p2p(pd);
                        lognorm1=log10(lognrnd(0,5));

                        p=exp(-dij./do(k));

                        for cj=1:10000
                            for ci=1:length(dij)
                                norm1=normrnd(0,1);
                                norm2=normrnd(0,1);
                                norm3=normrnd(0,1);


X1correlatedt(ci,cj)=s1*(sqrt(abs(p(ci)))*norm1+sqrt(1-
p(ci)*p(ci))*norm2);

X2correlatedt(ci,cj)=s2*(sqrt(abs(p(ci)))*norm1+sqrt(1-
p(ci)*p(ci))*norm3);
                            end
                        end


PLc1t=10*n*log10(4*pi*dti/lamda(k))+X1correlatedt;

PLc2t=10*n*log10(4*pi*dtj/lamda(k))+X2correlatedt;

                        mPLc1=sum(PLc1t')./10000;
```

```
                    mPLc2=sum(PLc2t')./10000;

                    qx = (b-(Pt(k) - mPLc1))/std_db;
                    P1(cx(cp), cy(cp)) = qfunc(qx);

                    qx = (b-(Pt(k) - mPLc2))/std_db;
                    P1(cx(cp-1), cy(cp-1)) = qfunc(qx);
                end
            end

            if debug==1
                fprintf(file_2, 'i=%d, j=%d\n', i,j);
                for w=1:N
                    fprintf(file_2, '%d ', P1(w,:));
                    fprintf(file_2, '\n');
                end
                fprintf(file_2, '\n');
            end


            %Insert into Probability Array temp array P1
            P(i,j,k) = {P1};
        end
    end
end

%Miss Probability
missj = ones(N,N);

x = repmat({[0 0]}, N, N);

%While Miss Probability is less than threshold
while (find(missj>T))


    %Initialize Gain Factor
    gik = zeros(N,N,K);
    % Initialize Miss Probability
    missj = ones(N,N);

    for k=1:K;
        for i=1:N;
            for j=1:N;
                if x{i,j}(1,2)==k
                    missj = missj .* (1 - P{i,j,k});
                end
            end
        end
    end

    % point
    point = ones(1,3);
    maximum = gik(1,1,1);
```

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

```
        %Gain Factor
        %For every type of sensor
        for i=1:N;
            %For every point
            for j=1:N;
                for k=1:K;
                   %if point is available
                    if x{i,j}(1,1) ~= 1
                        for xi=1:N;
                            for yj=1:N;
                                %Calculate Gain Factor
                                gik(i,j,k) = gik(i,j,k) +
P{xi,yj,k}(i,j)*(max(0, missj(xi,yj)- T));
                            end
                        end
                    end

                    if (gik(i,j,k)>maximum)
                       point = [i,j,k];
                       maximum = gik(i,j,k);
                    end

                end
            end
        end


        %Place Sensor
        x(point(1), point(2)) = {[1,point(3)]};

        missj = ones(N,N);

        %Update points remaining coverage
        for k=1:K;
            for i=1:N;
                for j=1:N;
                    if x{i,j}(1,2)==k
                        missj = missj .* (1 - P{i,j,k});
                    end
                end
            end
        end

        for i=1:N;
            for j=1:N;
                X(i,j) = x{i,j}(1,1);
                S(i,j) = x{i,j}(1,2);
            end
        end


        aa=1;

    end
```

UNDER REVIEW

```
    %Close Files Opened
    if debug==1
        fclose(file_1);
        fclose(file_2);
    end

    t(1,n) = toc;

    %Display Results
    for i=1:N;
        for j=1:N;
            X(i,j) = x{i,j}(1,1);
            S(i,j) = x{i,j}(1,2);
            if x{i,j}(1,1)==1;
                zg(1,n) = zg(1,n) + c(1,x{i,j}(1,2));
            end
        end
    end

    S
    zg
    t

end
```

ПАНЕЛЬНОЕ ПО

Κώδικας Matlab για επίλυση του γραμμικού προβλήματος για lognormal σκίαση.

```
%Solve the linear programming problem of ECP2
clear;
clc;

%grid dimensions
V = horzcat([4:15],[20]);
%Objective Values
Fval = zeros(1,length(V));

%miss probability threshold
T=0.01;

xl=10;
Pt = [10,10];
lamda = [0.33, 0.69]; %m
std_db = 5;
b=-15;
s1=1.778;
s2=1.778;
do = [10, 20];
n=2.0;

%sensors cost
c = [100 125];

%Number of Sensors types
K=2;

%debug=1 => create text files
debug=1;

%For sll grid dimensions
for n = 1:length(V)
    %Grid Dimension
    N = V(1,n);

    %Initialize Coverage Coefficient Matrice
    A = zeros(1,N*N*K);

    %Matrice Used for Constraint (3)
    G = zeros(1,N*N*K);

    %Objective function
    f = [];
    for k=1:K
        f = horzcat(f,repmat(c(1,k),1,N*N));
    end
```

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΑ

```
if debug==1
    file_1 = fopen('ecp4_ip_.lg4','w');
    fprintf(file_1, 'MIN=');

    for i=1:2*N*N
        fprintf(file_1, '%5.4f*x%d', f(1,i), i);
        if i<2*N*N
            fprintf(file_1, '+');
        end
    end

    fprintf(file_1, ';\n');
end

%constraint (2)
%For all types of sensors
for k=1:K;
    %Helper
    z=0;
    %For all greedy points
    for i=1:N;
        for j=1:N;

            %Coverage If Sensor is Places at point (i,j)
            A1=zeros(N,N);
            A2=zeros(N,N);

            %Sensor Location
            A2(i,j)=1;

            %%%%%%%%%%%%%%%

            Dist = zeros(N,N);
            p2p = zeros(1, 1);
            p=1;

            %Temporary Array - Propability
            A1=zeros(N,N);

            %Calculate probability if sensor is in (i,j)
            for xi=1:N;
                for yj=1:N;
                    eud = sqrt(((i*xl-xi*xl)^2)+((j*xl-yj*xl)^2));
                    Dist(xi,yj)=eud;

                    if (find(p2p==eud))
                        %
                    else
                        p2p(1, p) = eud;
                        p=p+1;
                    end
                end
            end
```

UNDER INVOICE

```
                A1(i, j) = 0.99;

                for pd=1:p-1
                    [cx,cy] = find(Dist==p2p(pd));

                    if length(cx)==1 && Dist(cx(1),cy(1))~=0 &&
p2p(pd)~=0
                        ceud = 0;
                        dij=[ceud];

                        dti=p2p(pd);
                        dtj=p2p(pd);
                        lognorm1=log10(lognrnd(0,5));

                        p=exp(-dij./do(k));

                        for cj=1:10000
                            for ci=1:length(dij)
                                norm1=normrnd(0,1);
                                norm2=normrnd(0,1);

X1correlatedt(ci,cj)=s1*(sqrt(abs(p(ci)))*norm1+sqrt(1-
p(ci)*p(ci))*norm2);
                            end
                        end

PLc1t=10*n*log10(4*pi*dti/lamda(k))+X1correlatedt;

                        mPLc1=sum(PLc1t')./10000;

                        qx = (b-(Pt(k) - mPLc1))/std_db;
                        A1(cx(1), cy(1)) = qfunc(qx);
                    end

                    for (cp=2:length(cx))
                        ceud = sqrt(((cx(cp)-cx(cp-1))^2)+((cy(cp)-
cy(cp-1))^2));
                        dij=[ceud];

                        dti=p2p(pd);
                        dtj=p2p(pd);
                        lognorm1=log10(lognrnd(0,5));

                        p=exp(-dij./do(k));

                        for cj=1:10000
                            for ci=1:length(dij)
                                norm1=normrnd(0,1);
                                norm2=normrnd(0,1);
                                norm3=normrnd(0,1);
```

```
X1correlatedt(ci,cj)=s1*(sqrt(abs(p(ci)))*norm1+sqrt(1-
p(ci)*p(ci))*norm2);

X2correlatedt(ci,cj)=s2*(sqrt(abs(p(ci)))*norm1+sqrt(1-
p(ci)*p(ci))*norm3);
                            end
                        end


PLc1t=10*n*log10(4*pi*dti/lamda(k))+X1correlatedt;

PLc2t=10*n*log10(4*pi*dtj/lamda(k))+X2correlatedt;

                        mPLc1=sum(PLc1t')./10000;
                        mPLc2=sum(PLc2t')./10000;

                        qx = (b-(Pt(k) - mPLc1))/std_db;
                        A1(cx(cp), cy(cp)) = -log(1-qfunc(qx));

                        qx = (b-(Pt(k) - mPLc2))/std_db;
                        A1(cx(cp-1), cy(cp-1)) = -log(1-qfunc(qx));
                    end
                end

                %%%%%%%%%%%%%%

                z=z+1;

                %Reshape Matrices for linprog
                A(z,N*N*(k-1)+1:N*N*k) = reshape(A1,1,N^2);
                G(z,N*N*(k-1)+1:N*N*k) = reshape(A2,1,N^2);

            end
        end
    end

    %Right part of inequality
    B = ones(1,K*N^2);
    B(1,1:N*N) = log(T);

    %Create lingo file
    if debug==1
        TX = [-A ; G];

        for i=1:2*N*N;
            for j=1:2*N*N;
                fprintf(file_1, '%5.4f*x%d', TX(i,j),j);
                if j<2*N*N
                    fprintf(file_1, '+');
                end
            end

            if i>N*N
```

```
                fprintf(file_1, '<=%5.4f;', B(1,i));
            else
                fprintf(file_1, '<=%5.4f;', B(1,i));
            end

            fprintf(file_1, '\n') ;
        end

        for i=1:2*N*N
            fprintf(file_1, '@BIN(x%d);', i);
            fprintf(file_1, '\n') ;
        end

        fclose(file_1);
    end

    %Right part of inequality
    A = [-A ; G];

    %Lower and Upper bounds
    lb=zeros(K*N^2,1);
    ub=ones(K*N^2,1);

    %Solve the linear programming problem
    [x, fval, exitflag, output]=linprog(f, A, B,[],[], lb, ub);

    %Update Results Metrice
    Fval(1,n) = fval;
end

Fval
```

UNEDITED COPY

.NET κώδικας για επίλυση αλγορίθμου greedy για τέλεια κάλυψη

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Diagnostics;
using System.IO;

namespace ConsoleAppication
{
    public class ecp1
    {
        private const byte K = 2;
        private const long b = 2;
        public int N;
        private Stopwatch stopWatch;
        private int debug = 0;
        private StreamWriter sw;

        public void Ecp1AppRun()
        {
            Console.Write("Enter N: ");
            string input = Console.ReadLine();

            try
            {
                N = Int32.Parse(input);
            }
            catch
            {
                N = 4;
            }

            Console.Write("Debug? (0:No, 1:Coverage, 2:Full): ");
            input = Console.ReadLine();

            try
            {
                debug = Int32.Parse(input);
                if (debug > 2) debug = 2;
            }
            catch { }

            if (debug != 0)
            {
                string path = @"certain" + N + "x" + N + ".txt";
                if (File.Exists(path))
                {
                    File.Delete(path);
                }

                FileStream file = new FileStream(path,
FileMode.CreateNew, FileAccess.ReadWrite);
                sw = new StreamWriter(file);
```

ΑΝΕΠΙΣΤΗΜΟΝΙΚΟ ΕΡΓΑΣΤΗΡΙΟ

```
            }

            stopWatch = new Stopwatch();
            stopWatch.Start();

            //Define index parameters
            int i, j, k, x, y;

            //ranges
            long[] d = new long[K] { 1, 2 };
            //costs
            double[] c = new double[K] { 100, 150 };
            //sensors
            long[,][,] s = new long[N, N][,];
            //covarage
            long[,] covj = new long[N, N];

            //Initialize Sensor Points
            for (i = 0; i < N; i++)
            {
                for (j = 0; j < N; j++)
                {
                    s[i, j] = new long[,] { { 0, 0 } };
                }
            }

            //a
            long[, ,][,] A = new long[N, N, K][,];
            //gik
            double[, ,] gik = new double[N, N, K];

            if (debug == 2) sw.Write("Coverage Coefficient");

            for (k = 0; k < K; k++)
            {
                for (i = 0; i < N; i++)
                {
                    for (j = 0; j < N; j++)
                    {
                        A[i, j, k] = new long[N, N];
                        if (debug == 2) Console.WriteLine("(" + i + ","
+ j + ")");

                        for (x = 0; x < N; x++)
                        {
                            for (y = 0; y < N; y++)
                            {
                                if (Math.Sqrt(Math.Pow(i - x, 2) +
Math.Pow(j - y, 2)) <= d[k]) { A[i, j, k][x, y] = 1; }
                                if (debug == 2) Console.Write(A[i, j,
k][x, y] + " ");
                            }
                            if (debug == 2) Console.WriteLine();
                        }

                        if (debug == 2) Console.WriteLine();
```

UNREGISTERED

```
                }
            }
        }

        long[,] cov = new long[N, N];
        ZerosCov(cov);
        double value = 0;

        while (Find(cov))
        {
            ZerosGik(gik);

            int[] maxgik =  new int[3] { 0, 0, 0 };
            double maximum = gik[0, 0, 0];


            for (i = 0; i < N; i++)
            {
                for (j = 0; j < N; j++)
                {
                    for (k = 0; k < K; k++)
                    {
                        if (s[i, j][0, 0] != 1)
                        {
                            for (x = 0; x < N; x++)
                            {
                                for (y = 0; y < N; y++)
                                {
                                    gik[i, j, k] = gik[i, j, k] +
A[x, y, k][i, j] * (GetMaxG(b, cov[x, y]) / c[k]);

                                }
                            }
                        }
                        if (gik[i, j, k] > maximum)
                        {
                            maximum = gik[i, j, k];
                            maxgik = new int[3] { i, j, k };
                        }
                    }
                }
            }

            s[maxgik[0], maxgik[1]][0, 0] = 1;
            s[maxgik[0], maxgik[1]][0, 1] = (long)maxgik[2];

            if (debug==2)
            {
                for (k = 0; k < K; k++)
                {
                    sw.WriteLine("k= " + k);
                    Console.WriteLine("k= " + k);
                    for (i = 0; i < N; i++)
                    {
                        for (j = 0; j < N; j++)
                        {
                            sw.Write(gik[i, j, k] + " ");
```

```
                                Console.Write(gik[i, j, k] + " ");
                            }
                            sw.WriteLine("");
                            Console.WriteLine("");
                        }
                        sw.WriteLine("");
                        Console.WriteLine("");
                        Console.ReadKey();
                    }
                }

                value = 0;
                ZerosCov(cov);

                //Calculate Coverage
                for (i = 0; i < N; i++)
                {
                    for (j = 0; j < N; j++)
                    {
                        if (s[i, j][0, 0] == 1)
                        {
                            cov = CopylongArrays(A[i, j, s[i, j][0,
1]], cov);
                        }
                    }
                }

                if (debug > 0)
                {
                    sw.WriteLine("Point: (" + (maxgik[0] + 1) + "," +
(maxgik[1] + 1) + ") - Sensor Type:" + (maxgik[2] + 1));
                }


                for (i = 0; i < N; i++)
                {
                    for (j = 0; j < N; j++)
                    {
                        if (s[i, j][0, 0] == 1)
                        {
                            if (debug > 0) sw.Write((s[i, j][0, 1] + 1)
+ " ");
                            value = value + c[s[i, j][0, 1]];
                        }
                        else
                        {
                            if (debug > 0) sw.Write((s[i, j][0, 1]) + "
");
                        }
                    }

                    if (debug > 0)
                    {
                        sw.Write("   ");

                        for (j = 0; j < N; j++)
                        {
```

UNDER THE SEA

```
                            sw.Write(cov[i, j] + " ");
                        }

                        sw.WriteLine();
                    }
                }
            }

            stopWatch.Stop();
            if (debug > 0)  sw.Close();
            Console.WriteLine("Elapsed: {0}", stopWatch.Elapsed);
            Console.WriteLine("In Sec: {0}",
stopWatch.ElapsedMilliseconds);

            Console.WriteLine(value);

            int ktot = 0, k1 = 0, k2 = 0;

            for (i = 0; i < N; i++)
            {
                for (j = 0; j < N; j++)
                {
                    if (s[i, j][0, 0] == 1)
                    {
                        ktot++;
                        if (s[i, j][0, 1] == 0) k1++;
                        else k2++;
                    }
                }
            }

            Console.WriteLine("k1=" + k1 + ", k2=" + k2 + ", ktot=" +
ktot);
            Console.ReadKey();
        }

        static long GetMaxG(long b, long cov)
        {
            long diff = (b - cov);

            if ((diff) < 0)
            {
                diff = 0;
            }

            return diff;
        }

        private int[] max(double[, ,] t)
        {
            int[] point = new int[3] { 0, 0, 0 };
            double maximum = t[0, 0, 0];

            for (int k = K-1; k >= 0; k--)
            {
                for (int i = 0; i < N; i++)
                {
```

```
            for (int j = 0; j < N; j++)
            {
                if (t[i, j, k] > maximum)
                {
                    point = new int[3] { i, j, k };
                    maximum = t[i, j, k];
                }
            }
        }
    }
    return point;
}

private long[,] CopylongArrays(long[,] From, long[,] To)
{
    long[,] Res = new long[this.N, this.N];

    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            Res[i, j] = (long)(To[i, j] + From[i, j]);
        }
    }

    return Res;
}

private void ZerosCov(long[,] cov)
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            cov[i, j] = 0;
        }
    }
}

private void ZerosGik(double[, ,] gik)
{
    int k, i, j;

    for (k = 0; k < K; k++)
    {
        for (i = 0; i < N; i++)
        {
            for (j = 0; j < N; j++)
            {
                gik[i, j, k] = 0;
            }
        }
    }
}

private bool Find(long[,] cov)
{
```

ОЗНАЙОМЧИЙ ПРИМІРНИК

```
            bool res = false;

            for (int i = 0; i < N; i++)
            {
                for (int j = 0; j < N; j++)
                {
                    if (cov[i, j] < b)
                    {
                        res = true;
                        j = i = N;
                    }
                }
            }

            return res;
        }
    }
}
```

ЛАБЕЛЛИНО ПЕРАК

.NET κώδικας για επίλυση αλγορίθμου greedy για αβέβαιη κάλυψη

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Diagnostics;
using System.IO;

namespace ConsoleAppication
{
    public class ecp2
    {
        private const byte K = 2;
        public int N;
        private Stopwatch stopWatch;
        private int debug = 0;
        private StreamWriter sw;
        private const double T = 0.01D;

        public void Ecp2AppRun()
        {
            Console.Write("Enter N: ");
            string input = Console.ReadLine();

            try
            {
                N = Int32.Parse(input);
            }
            catch
            {
                N = 4;
            }

            Console.Write("Debug? (0:No, 1:Coverage, 2:Full): ");
            input = Console.ReadLine();

            try
            {
                debug = Int32.Parse(input);
                if (debug > 2) debug = 2;
            }
            catch { }

            if (debug != 0)
            {
                string path = @"uncertain" + N + "x" + N + ".txt";
                if (File.Exists(path))
                {
                    File.Delete(path);
                }

                FileStream file = new FileStream(path,
FileMode.CreateNew, FileAccess.ReadWrite);
                sw = new StreamWriter(file);
            }
```

UNIVERSITY OF PIRAEUS

```
            stopWatch = new Stopwatch();
            stopWatch.Start();

            //Define index parameters
            int i, j, k, x, y;

            double[] a = new double[K] { 0.6D, 0.5D };
            //ranges
            long[] d = new long[K] { 1, 2 };
            //costs
            double[] c = new double[K] { 100, 125 };
            //sensors
            long[,][,] s = new long[N, N][,];

            //Initialize Sensor Points
            for (i = 0; i < N; i++)
            {
                for (j = 0; j < N; j++)
                {
                    s[i, j] = new long[,] { { 0, 0 } };
                }
            }

            //P
            double[, ,][,] P = new double[N, N, K][,];
            //gik
            double[, ,] gik = new double[N, N, K];
            //eud
            double eud = 0;

            if (debug == 2) sw.Write("Coverage Coefficient");

            for (k = 0; k < K; k++)
            {
                for (i = 0; i < N; i++)
                {
                    for (j = 0; j < N; j++)
                    {
                        P[i, j, k] = new double[N, N];
                        if (debug==2) sw.WriteLine("(" + i + "," + j +
")");

                        for (x = 0; x < N; x++)
                        {
                            for (y = 0; y < N; y++)
                            {
                                eud = Math.Sqrt(Math.Pow(i - x, 2) +
Math.Pow(j - y, 2));

                                if (eud > 0)
                                {
                                    P[i, j, k][x, y] =
(float)Math.Exp(-a[k] * eud);
                                }
                                else
                                {
                                    P[i, j, k][x, y] = 0.99;
```

UNREGISTERED

```
                                    }
                                    if (debug == 2) sw.Write(P[i, j, k][x,
y] + " ");
                                }
                                if (debug == 2) sw.WriteLine();
                            }

                            if (debug == 2) sw.WriteLine();
                        }
                    }
                }

                double[,] missj = new double[N, N];
                OnesMissj(missj);
                double value = 0;

                while (Find(missj))
                {
                    ZerosGik(gik);

                    int[] maxgik =  new int[3] { 0, 0, 0 };
                    double maximum = gik[0, 0, 0];

                    for (i = 0; i < N; i++)
                    {
                        for (j = 0; j < N; j++)
                        {
                            for (k = 0; k < K; k++)
                            {
                                if (s[i, j][0, 0] != 1)
                                {
                                    for (x = 0; x < N; x++)
                                    {
                                        for (y = 0; y < N; y++)
                                        {
                                            gik[i, j, k] = gik[i, j, k] +
(P[x, y, k][i, j] * GetMaxG(missj[x, y] - T));
                                        }
                                    }
                                }
                                if (gik[i, j, k] > maximum)
                                {
                                    maximum = gik[i, j, k];
                                    maxgik = new int[3] { i, j, k };
                                }
                            }
                        }
                    }

                    s[maxgik[0], maxgik[1]][0, 0] = 1;
                    s[maxgik[0], maxgik[1]][0, 1] = (long)maxgik[2];

                    if (debug==2)
                    {
                        for (k = 0; k < K; k++)
                        {
```

```
                        sw.WriteLine("k= " + k);
                        for (i = 0; i < N; i++)
                        {
                            for (j = 0; j < N; j++)
                            {
                                sw.Write(gik[i, j, k] + " ");
                            }
                            sw.WriteLine("");
                        }
                        sw.WriteLine("");
                    }
                }

                value = 0;
                OnesMissj(missj);

                //Calculate Coverage
                for (i = 0; i < N; i++)
                {
                    for (j = 0; j < N; j++)
                    {
                        if (s[i, j][0, 0] == 1)
                        {
                            missj = MultiplyDoubleArrays(P[i, j, s[i,
j][0, 1]], missj);
                        }
                    }
                }

                if (debug > 0)
                {
                    sw.WriteLine("Point: (" + (maxgik[0] + 1) + "," +
(maxgik[1] + 1) + ") - Sensor Type:" + (maxgik[2] + 1));
                }

                for (i = 0; i < N; i++)
                {
                    for (j = 0; j < N; j++)
                    {
                        if (s[i, j][0, 0] == 1)
                        {
                            if (debug > 0) sw.Write((s[i, j][0, 1] + 1)
+ " ");
                            value = value + c[s[i, j][0, 1]];
                        }
                        else
                        {
                            if (debug > 0) sw.Write((s[i, j][0, 1]) + "
");
                        }
                    }

                    if (debug > 0)
                    {
                        sw.Write("   ");

                        for (j = 0; j < N; j++)
```

```
                        {
                            sw.Write(missj[i, j] + " ");
                        }

                        sw.WriteLine();
                    }
                }
            }


            stopWatch.Stop();
            if (debug > 0)  sw.Close();
            Console.WriteLine("Elapsed: {0}", stopWatch.Elapsed);
            Console.WriteLine("In milliseconds: {0}",
stopWatch.ElapsedMilliseconds);

            int ktot = 0, k1 = 0, k2 = 0;

            for (i = 0; i < N; i++)
            {
                for (j = 0; j < N; j++)
                {
                    if (s[i, j][0, 0] == 1)
                    {
                        ktot++;
                        if (s[i, j][0, 1] == 0) k1++;
                        else k2++;
                    }
                }
            }

            Console.WriteLine("k1=" + k1 + ", k2=" + k2 + ", ktot=" +
ktot);
            Console.WriteLine(value);
            Console.ReadKey();
        }

        private double GetMaxG(double missj)
        {
            if ((missj) < 0)
            {
                missj = 0;
            }

            return missj;
        }

        private int[] max(double[, ,] t)
        {
            int[] point = new int[3] { 0, 0, 0 };
            double maximum = t[0, 0, 0];

            for (int k = K-1; k >= 0; k--)
            {
                for (int i = 0; i < N; i++)
                {
                    for (int j = 0; j < N; j++)
```

UNREGISTERED

```
                    {
                        if (t[i, j, k] > maximum)
                        {
                            point = new int[3] { i, j, k };
                            maximum = t[i, j, k];
                        }
                    }
                }
            }
            return point;
        }

        private double[,] MultiplyDoubleArrays(double[,] From,
double[,] To)
        {
            double[,] Res = new double[this.N, this.N];

            for (int i = 0; i < N; i++)
            {
                for (int j = 0; j < N; j++)
                {
                    Res[i, j] = (double)(To[i, j] * (1 - From[i, j]));
                }
            }

            return Res;
        }

        private void ZerosCov(long[,] cov)
        {
            for (int i = 0; i < N; i++)
            {
                for (int j = 0; j < N; j++)
                {
                    cov[i, j] = 0;
                }
            }
        }

        private void ZerosGik(double[, ,] gik)
        {
            int k, i, j;

            for (k = 0; k < K; k++)
            {
                for (i = 0; i < N; i++)
                {
                    for (j = 0; j < N; j++)
                    {
                        gik[i, j, k] = 0;
                    }
                }
            }
        }

        private bool Find(double[,] missj)
        {
```

ОЗНАЙОМЧИЙ ПРИМІРНИК

```
            bool res = false;

            for (int i = 0; i < N; i++)
            {
                for (int j = 0; j < N; j++)
                {
                    if (missj[i, j] > T)
                    {
                        res = true;
                        j = i = N;
                    }
                }
            }

            return res;
        }

        private void OnesMissj(double[,] missj)
        {
            for (int i = 0; i < N; i++)
            {
                for (int j = 0; j < N; j++)
                {
                    missj[i, j] = 1;
                }
            }
        }
    }
}
```

ЛАБОРАТОРИЯ

.NET κώδικας για επίλυση αλγορίθμου greedy για αβέβαιη κάλυψη με περιορισμένη ακτίνα

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Diagnostics;
using System.IO;

namespace ConsoleAppication
{
    public class ecp3
    {
        private const byte K = 2;
        public int N;
        private Stopwatch stopWatch;
        private int debug = 0;
        private StreamWriter sw;
        private const double T = 0.01D;

        public void Ecp3AppRun()
        {
            Console.Write("Enter N: ");
            string input = Console.ReadLine();

            try
            {
                N = Int32.Parse(input);
            }
            catch
            {
                N = 4;
            }

            Console.Write("Debug? (0:No, 1:Coverage, 2:Full): ");
            input = Console.ReadLine();

            try
            {
                debug = Int32.Parse(input);
                if (debug > 2) debug = 2;
            }
            catch { }

            if (debug != 0)
            {
                string path = @"ecp3" + N + "x" + N + ".txt";
                if (File.Exists(path))
                {
                    File.Delete(path);
                }

                FileStream file = new FileStream(path,
FileMode.CreateNew, FileAccess.ReadWrite);
```

```
            sw = new StreamWriter(file);
        }

            stopWatch = new Stopwatch();
            stopWatch.Start();

            //Define index parameters
            int i, j, k, x, y;

            //a
            double a;
            //lamda
            double[] l = new double[K] { 0.5D, 0.6D };
            //bita
            double[] b = new double[K] { 0.5D, 0.6D };
            //costs
            double[] c = new double[K] { 100, 125 };
            //sensors
            long[,][,] s = new long[N, N][,];
            //r
            double[] r = new double[K] { 1, 2 };
            //re
            double[] re = new double[K] { 0.8D, 1.6D };


            //Initialize Sensor Points
            for (i = 0; i < N; i++)
            {
                for (j = 0; j < N; j++)
                {
                    s[i, j] = new long[,] { { 0, 0 } };
                }
            }

            //P
            double[, ,][,] P = new double[N, N, K][,];
            //gik
            double[, ,] gik = new double[N, N, K];
            //eud
            double eud = 0, rre_add = 0, rre_diff = 0;

            if (debug == 2) sw.Write("Coverage Coefficient");

            for (k = 0; k < K; k++)
            {
                for (i = 0; i < N; i++)
                {
                    for (j = 0; j < N; j++)
                    {
                        P[i, j, k] = new double[N, N];
                        if (debug == 2) Console.WriteLine("(" + i +
","" + j + ")");


                        for (x = 0; x < N; x++)
                        {
                            for (y = 0; y < N; y++)
                            {
```

```
                                    eud = Math.Sqrt(Math.Pow(i - x, 2)
+ Math.Pow(j - y, 2));
                                    rre_add = r[k] + re[k];
                                    rre_diff = r[k] - re[k];

                                    if (rre_add <= eud)
                                    {
                                        P[i, j, k][x, y] = 0;
                                    }
                                    else if ((rre_diff < eud) && (eud <
rre_add))
                                    {
                                        a = eud - rre_diff;
                                        P[i, j, k][x, y] =
(double)Math.Exp(-l[k] * Math.Pow(a, b[k]));
                                    }
                                    else if (rre_diff >= eud)
                                    {
                                        P[i, j, k][x, y] = 0.99;
                                    }

                                    if (debug == 2) Console.Write(P[i,
j, k][x, y] + " ");

                                }
                                if (debug == 2) Console.WriteLine();
                            }

                            if (debug == 2)
                            {
                                Console.WriteLine();
                                Console.ReadKey();
                            }
                        }
                    }
                }

                double[,] missj = new double[N, N];
                OnesMissj(missj);
                double value = 0;

                while (Find(missj))
                {
                    ZerosGik(gik);

                    int[] maxgik = new int[3] { 0, 0, 0 };
                    double maximum = gik[0, 0, 0];

                    for (i = 0; i < N; i++)
                    {
                        for (j = 0; j < N; j++)
                        {
                            for (k = 0; k < K; k++)
                            {
                                if (s[i, j][0, 0] != 1)
                                {
                                    for (x = 0; x < N; x++)
```

```
                                    {
                                        for (y = 0; y < N; y++)
                                        {
                                            gik[i, j, k] = gik[i, j, k]
+ (P[x, y, k][i, j] * GetMaxG(missj[x, y] - T));
                                        }
                                    }
                                }
                                if (gik[i, j, k] > maximum)
                                {
                                    maximum = gik[i, j, k];
                                    maxgik = new int[3] { i, j, k };
                                }
                            }
                        }
                    }

                    s[maxgik[0], maxgik[1]][0, 0] = 1;
                    s[maxgik[0], maxgik[1]][0, 1] = (long)maxgik[2];

                    if (debug == 2)
                    {
                        for (k = 0; k < K; k++)
                        {
                            Console.WriteLine("k= " + k);
                            for (i = 0; i < N; i++)
                            {
                                for (j = 0; j < N; j++)
                                {
                                    Console.Write(gik[i, j, k] + " ");
                                }
                                Console.WriteLine("");
                            }
                            Console.WriteLine("");
                            Console.ReadKey();
                        }
                    }

                    value = 0;
                    OnesMissj(missj);

                    //Calculate Coverage
                    for (i = 0; i < N; i++)
                    {
                        for (j = 0; j < N; j++)
                        {
                            if (s[i, j][0, 0] == 1)
                            {
                                missj = MultiplyDoubleArrays(P[i, j,
s[i, j][0, 1]], missj);
                            }
                        }
                    }

                    if (debug > 0)
                    {
```

UNDER PEER REVIEW

```
                    Console.WriteLine("Point: (" + (maxgik[0] + 1)
+ "," + (maxgik[1] + 1) + ") - Sensor Type:" + (maxgik[2] + 1));
                }

                for (i = 0; i < N; i++)
                {
                    for (j = 0; j < N; j++)
                    {
                        if (s[i, j][0, 0] == 1)
                        {
                            if (debug > 0) sw.Write((s[i, j][0, 1]
+ 1) + " ");

                            value = value + c[s[i, j][0, 1]];
                        }
                        else
                        {
                            if (debug > 0) sw.Write((s[i, j][0, 1])
+ " ");

                        }
                    }

                    if (debug > 0)
                    {
                        sw.Write("   ");

                        for (j = 0; j < N; j++)
                        {
                            sw.Write(missj[i, j] + " ");
                        }

                        sw.WriteLine();
                    }
                }
            }


            stopWatch.Stop();
            if (debug > 0) sw.Close();
            Console.WriteLine("Elapsed: {0}", stopWatch.Elapsed);
            Console.WriteLine("In milliseconds: {0}",
stopWatch.ElapsedMilliseconds);

            int ktot = 0, k1 = 0, k2 = 0;

            for (i = 0; i < N; i++)
            {
                for (j = 0; j < N; j++)
                {
                    if (s[i, j][0, 0] == 1)
                    {
                        ktot++;
                        if (s[i, j][0, 1] == 0) k1++;
                        else k2++;
                    }
                }
            }
```

```
                Console.WriteLine("k1=" + k1 + ", k2=" + k2 + ", ktot="
+ ktot);
                Console.WriteLine(value);
                Console.ReadLine();
            }


        private double GetMaxG(double missj)
        {
            if ((missj) < 0)
            {
                missj = 0;
            }

            return missj;
        }

        private int[] max(double[, ,] t)
        {
            int[] point = new int[3] { 0, 0, 0 };
            double maximum = t[0, 0, 0];

            for (int k = K-1; k >= 0; k--)
            {
                for (int i = 0; i < N; i++)
                {
                    for (int j = 0; j < N; j++)
                    {
                        if (t[i, j, k] > maximum)
                        {
                            point = new int[3] { i, j, k };
                            maximum = t[i, j, k];
                        }
                    }
                }
            }
            return point;
        }

        private double[,] MultiplyDoubleArrays(double[,] From,
double[,] To)
        {
            double[,] Res = new double[this.N, this.N];

            for (int i = 0; i < N; i++)
            {
                for (int j = 0; j < N; j++)
                {
                    Res[i, j] = (double)(To[i, j] * (1 - From[i, j]));
                }
            }

            return Res;
        }

        private void ZerosCov(long[,] cov)
        {
```

```
            for (int i = 0; i < N; i++)
            {
                for (int j = 0; j < N; j++)
                {
                    cov[i, j] = 0;
                }
            }
        }

        private void ZerosGik(double[, ,] gik)
        {
            int k, i, j;

            for (k = 0; k < K; k++)
            {
                for (i = 0; i < N; i++)
                {
                    for (j = 0; j < N; j++)
                    {
                        gik[i, j, k] = 0;
                    }
                }
            }
        }

        private bool Find(double[,] missj)
        {
            bool res = false;

            for (int i = 0; i < N; i++)
            {
                for (int j = 0; j < N; j++)
                {
                    if (missj[i, j] > T)
                    {
                        res = true;
                        j = i = N;
                    }
                }
            }

            return res;
        }

        private void OnesMissj(double[,] missj)
        {
            for (int i = 0; i < N; i++)
            {
                for (int j = 0; j < N; j++)
                {
                    missj[i, j] = 1;
                }
            }
        }
    }
}
```

ENGINEERING