ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ, ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

ΑΚΑΔΗΜΑΪΚΟ ΕΤΟΣ 2011-2012

Μελέτη και αποτίμηση αλγορίθμων επίλυσης του προβλήματος ανάθεσης

Hungarian Algorithm and other approaches for the assignment problem

Όλγα Μανιάτη

Διπλωματική εργασία

Όλγα Μανιάτη
Τμήμα Μηχανικών Η/Υ, Τηλεπικοινωνιών και Δικτύων
Πολυτεχνική Σχολή του Πανεπιστημίου Θεσσαλίας

Επιβλέπων καθηγητής:

Ακρίτας Αλκιβιάδης
Αναπληρωτής Καθηγητής
Μαθηματικό Τμήμα
Σχολή Θετικών Επιστημών του Πανεπιστήμιου Αθηνών

Διμελής εξεταστική επιτροπή:

Σταμούλης Γεώργιος
Καθηγητής
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
ΕΜΠ

Ακρίτας Αλκιβιάδης

Αφιερωμένο στην οικογένεια μου

που πάντα με στήριζε

Περιεχόμενα

1. Εισαγωγή

Στην παρούσα διπλωματική εργασία μελετώνται τρόποι επίλυσης του προβλήματος της ανάθεσης. Το πρόβλημα της ανάθεσης αφορά την αντιστοίχηση m δραστηριοτήτων σε m πόρους έτσι ώστε κάθε ένας πόρος να αντιστοιχηθεί σε μία ακριβώς δραστηριότητα και το αντίθετο. Η αντιστοίχηση του πόρου i στην δραστηριότητα j εισάγει κόστος ίσο με c_{ij}. Ένα ενδιαφέρον θέμα βελτιστοποίησης είναι να βρούμε την αντιστοίχηση με το μικρότερο συνολικό κόστος. Το πρόβλημα αυτό είναι δύσκολο να λυθεί λόγο της διακριτής του φύσης και του μεγάλου πλήθους των διαφορετικών αντιστοιχήσεων που υπάρχουν και πρέπει να ελεγχθούν μία προς μία.

Αρχικά περιγράφουμε τους τρόπους επίλυσης του προβλήματος που έχουν προταθεί στο παρελθόν, δίνοντας έμφαση στον Ουγγρικό αλγόριθμο. Ο Ουγγρικός αλγόριθμος λύνει το παραπάνω πρόβλημα με βέλτιστο τρόπο και πολυπλοκότητα κυβική ως προς το μέγεθος της εισόδου. Η υψηλή πολυπλοκότητα εκτέλεσης του Hungarian αλγορίθμου έχει ως αποτέλεσμα να μην μπορεί να εφαρμοστεί ο εν λόγω αλγόριθμος σε εφαρμογές που απαιτούν γρήγορες αποφάσεις. Για αυτό το λόγο, στη συνέχεια προτείνουμε ένα νέο αλγόριθμο, ο οποίος λειτουργεί με ευριστικό τρόπο, πετυχαίνοντας μικρότερη πολυπλοκότητα εκτέλεσης. Τέλος, παρουσιάζουμε αποτελέσματα προσομοιώσεων που συγκρίνουν την απόδοση του προτεινόμενου αλγορίθμου για διάφορα στιγμιότυπα εισόδου του προβλήματος.

2. Το πρόβλημα ανάθεσης

Το πρόβλημα ανάθεσης ή αντιστοίχησης (αγγλ. assignment problem, γαλλ. Problème d'affectation) αφορά την κατανομή m πόρων σε m ακριβώς δραστηριότητες έτσι ώστε να βελτιστοποιείται το προκύπτον αποτέλεσμα, με τον περιορισμό ότι κάθε πόρος μπορεί να χρησιμοποιηθεί μόνο σε μια δραστηριότητα και η κάθε δραστηριότητα να χρησιμοποιήσει μόνον έναν πόρο.

Το πρόβλημα ανάθεσης τίθεται ως πρόβλημα ελαχιστοποίησης κόστους. Για κάθε ανάθεση ενός πόρου i, i=1,2, ...,m σε καθεμία από τις δραστηριότητες j, j=1,2, ...,m, ορίζεται ένα κόστος c_{ij} και επιδιώκεται εκείνη η αντιστοίχιση των πόρων στις δραστηριότητες που ελαχιστοποιεί το συνολικό κόστος.

Ως πρόβλημα ανάθεσης μπορούν να μοντελοποιηθούν πολλά προβλήματα αποφάσεων, όπως η ανάθεση αποστολών σε εμπορικούς αντιπροσώπους, η εκτέλεση έργων από κατασκευάστριες εταιρείες, η καταχώρηση διαφημιστικών μηνυμάτων σε μέσα μαζικής ενημέρωσης, η ανάθεση δρομολογίων σε αεροσκάφη ή πλοία, η επεξεργασία μετάλλων από εναλλακτικούς τεχνολογικούς εξοπλισμούς, η επεξεργασία αποβλήτων από εναλλακτικά συστήματα προστασίας, η εγκατάσταση μονάδων κοινής ωφέλειας σε διαφορετικές περιοχές και περιφέρειες, κτλ. με στόχο την βελτιστοποίηση ενός ή περισσοτέρων οικονομικών, κοινωνικών, αναπτυξιακών, πολιτικών και άλλων δεικτών.

2.1 Μοντελοποίηση

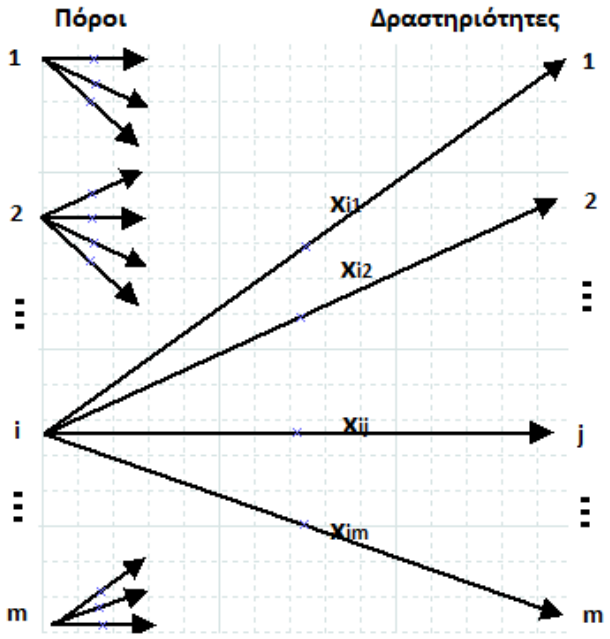
Το πρόβλημα ανάθεσης αναφέρεται στην «ένα - προς - ένα» αντιστοίχιση των στοιχείων ενός συνόλου οντοτήτων $A=\{a_1, a_2, \ldots, a_n\}$ στα στοιχεία ενός άλλου συνόλου οντοτήτων με τον ίδιο πληθικό αριθμό $B=\{b_1, b_2, \ldots, b_n\}$, κατά τρόπο που να βελτιστοποιείται ένα κριτήριο απόδοσης. Το κριτήριο απόδοσης διαμορφώνεται με βάση το κόστος ή το όφελος κάθε επιμέρους αντιστοίχισης, η οποία δίδεται ως παράμετρος του προβλήματος.

Χαρακτηριστικά παραδείγματα τέτοιων προβλημάτων είναι η ανάθεση εργασιών σε μηχανές ή εξυπηρετητές με το ελάχιστο κόστος ή η ανάθεση προσώπων σε θέσεις εργασίας με το μέγιστο όφελος. Στη γενική του μορφή το πρόβλημα ανάθεσης, διατυπωμένο ως πρόβλημα ελαχιστοποίησης, συνοψίζεται στον ακόλουθο πίνακα:

	b_1	**b_2**	**…**	**b_n**
a_1	c11	c12	…	c1n
a_2	c21	c22	…	c2n
…	…	…	…	…
a_n	cn1	cn2	…	cnn

όπου c_{ij} είναι το κόστος ανάθεσης της εργασίας i στον εξυπηρετητή j. Σύμφωνα με τον κανόνα της «ένα – προς - ένα» αντιστοίχισης και από το γεγονός ότι τα δύο σύνολα οντοτήτων έχουν το ίδιο πληθικό αριθμό, κάθε εργασία θα ανατεθεί σε έναν ακριβώς εξυπηρετητή και κάθε εξυπηρετητής θα αναλάβει μια ακριβώς εργασία.

Ταυτίζοντας εννοιολογικά το σύνολο Α με σημεία αναχώρησης και το σύνολο Β με προορισμούς, το πρόβλημα ανάθεσης είναι μια ειδική περίπτωση του κλασικού υποδείγματος του προβλήματος μεταφοράς. Τα ιδιαίτερα χαρακτηριστικά είναι ότι τα σημεία αναχώρησης είναι όσα ακριβώς και τα σημεία προορισμού ενώ η προσφορά σε κάθε σημείο αναχώρησης και η ζήτηση σε κάθε προορισμό είναι **ακριβώς** μια μονάδα προϊόντος.



Για την μοντελοποίηση του προβλήματος ανάθεσης, εισάγεται για κάθε δυνατή ανάθεση (i, j) μια μεταβλητή x_{ij}, η οποία περιορίζεται να λαμβάνει μόνο τις τιμές 0 ή 1 ($x_{ij} \in \{0,1\}$)

Αν $x_{ij} = 1$, τότε η εργασία i ανατίθεται στον εξυπηρετητή j.

Αν $x_{ij} = 0$, τότε η εργασία i δεν ανατίθεται στον εξυπηρετητή j.

Επομένως το εν λόγω πρόβλημα είναι ένα πρόβλημα διακριτού προγραμματισμού. Επομένως δεν μπορούν να χρησιμοποιηθούν τεχνικές όπως ο αλγόριθμος simplex (γραμμικός προγραμματισμός) ή πολλαπλασιαστές Lagrange (κυρτός προγραμματισμός) που θα έλυναν το πρόβλημα εύκολα και γρήγορα.

Σύμφωνα με τα παραπάνω, το πρόβλημα ανάθεσης μοντελοποιείται με το ακόλουθο ως πρόβλημα 0-1 ακέραιου γ.π.:

$$\min f = \sum_{i=1}^{m} \sum_{j=1}^{m} c_{ij} x_{ij}$$

$$\upsilon.\pi.$$

$$\sum_{i=1}^{m} x_{ij} = 1 \,,\; j = 1,\ldots, m$$

$$\sum_{j=1}^{m} x_{ij} = 1 \,, i = 1,\ldots, m$$

$$x_{ij} \in \{0,1\}$$

Όπως συμβαίνει και στο πρόβλημα μεταφοράς, ο πίνακας Α των συντελεστών των περιορισμών στο γ.π. του προβλήματος ανάθεσης αποτελείται από μοναδιαία μόνο στοιχεία, εμφανίζει συμμετρικότητα και είναι αραιός, έχει δηλαδή τα χαρακτηριστικά των προβλημάτων δικτύου.

Αντικαθιστώντας τον περιορισμό $x_{ij} \in \{0,1\}$ με τον περιορισμό μη αρνητικότητας $x_{ij} \geq 0$, το πρόβλημα μπορεί να επιλυθεί ως κλασικό γ.π. με τη μέθοδο simplex. Όμως, η μέθοδος simplex δεν μπορεί να δώσει τη βέλτιστη λύση, καθώς δεν εγγυάται ότι θα βρει ακέραια λύση, αλλά μπορεί να βρει μια λύση με κλασματικό μέρος. Μπορεί εύκολα να δειχτεί ότι αν η λύση περιέχει κλασματικό μέρος τότε η τιμή της λύσης θα είναι ένα κάτω όριο στην λύση του αρχικού προβλήματος ακέραιου προγραμματισμού. Αν όμως τύχει να βρει ακέραια λύση τότε αυτή θα είναι και βέλτιστη. Λόγω των παραπάνω χαρακτηριστικών του πίνακα, η βέλτιστη λύση θα είναι ακέραια και μάλιστα οι μεταβλητές θα λαμβάνουν τιμές από το σύνολο {0,1}.

2.2 Ουγγρικός αλγόριθμος

Το πρόβλημα ανάθεσης μπορεί να επιλυθεί με τον αλγόριθμο του **Kuhn** (γνωστό και ως Ουγγρικό αλγόριθμο). Αποτελεί μία αποτελεσματική και ταχεία μέθοδο προσδιορισμού της βέλτιστης ανάθεσης η οποία είναι βασισμένη στο θεώρημα του König. Ο αλγόριθμος βρίσκει την βέλτιστη λύση του γραμμικού προβλήματος ανάθεσης σε πολυωνυμικό χρόνο στην τάξη του $O(n^3)$.

Περιγραφή του αλγορίθμου ως ένας primal-dual αλγόριθμος

Στην προηγούμενη ενότητα ορίσαμε μαθηματικά το ακέραιο πρόβλημα της ανάθεσης. Τώρα, περιγράφουμε το γραμμικό πρόβλημα της ανάθεσης (P) το οποίο προκύπτει από το ακέραιο, αν πετάξουμε τους περιορισμούς ακεραιότητας:

$$
\begin{aligned}
& \text{Min} \quad \sum_{i,j} c_{ij} x_{ij} \\
& \text{subject to:} \\
(P) \qquad & \sum_{j} x_{ij} = 1 && i \in A \\
& \sum_{i} x_{ij} = 1 && j \in B \\
& x_{ij} \geq 0 && i \in A, j \in B.
\end{aligned}
$$

Η λύση του (P) θα είναι ένα κάτω όριο της λύσης του αρχικού ακέραιου προβλήματος, επειδή το σύνολο εφικτών λύσεων του (P) προβλήματος είναι ένα υπερσύνολο των εφικτών λύσεων του αρχικού ακέραιου προβλήματος.

Στη συνέχεια θα αποδείξουμε αλγοριθμικά τον Ουγγρικό αλγόριθμο. Ο αλγόριθμος αυτός ανήκει στην κατηγορία των primal-dual αλγορίθμων. Για να εξηγήσουμε τι σημαίνει αυτό, πρέπει πρώτα να ορίσουμε την έννοια της δυϊκότητας των γραμμικών προβλημάτων και συγκεκριμένα για το πρόβλημα (P) ορίζουμε μια μεταβλητή u_i για κάθε κόμβο i του συνόλου A και μία μεταβλητή v_j για κάθε κόμβο j του συνόλου B, έτσι ώστε $u_i + v_j <= c_{ij}$. Τότε το dual πρόβλημα ορίζεται ως εξής:

Οι περιορισμοί μπορούν να αλλάξουν σε: $\mathbf{w_{ij} \geq 0}$, όπου $\mathbf{w_{ij} = c_{ij} - u_i - v_j}$.

$$\begin{aligned} & \text{Max} \quad \sum_{i \in A} u_i + \sum_{j \in B} v_j \\ & \text{subject to:} \\ (D) \qquad & u_i + v_j \leq c_{ij} \qquad\qquad i \in A, j \in B. \end{aligned}$$

Αν βρω μία εφικτή λύση του (D) και μία λύση του αρχικού ακέραιου προβλήματος ανάθεσης που πετυχαίνουν την ίδια τιμή τότε η λύση του αρχικού ακέραιου προβλήματος ανάθεσης είναι **βέλτιστη**.

Ο Ουγγρικός αλγόριθμος εκτελεί μια σειρά από επαναλήψεις. Διαρκώς διατηρεί μια εφικτή λύση του dual προβλήματος (u,v,w) και προσπαθεί να βρει μια λύση του αρχικού προβλήματος που αναθέτει τον κόμβο i στον κόμβο j μόνο αν $\mathbf{w_{ij} = 0}$ (αν συμπεριλάμβανε κι άλλες αναθέσεις που δεν ικανοποιούν τον τελευταίο περιορισμό τότε είναι σίγουρο ότι η τιμή που πετύχαινε λύση του αρχικού προβλήματος θα ήταν χειρότερη από τη λύση του dual προβλήματος). Συγκεκριμένα ξεκινά με τη συγκεκριμένη λύση του dual προβλήματος $\mathbf{u_i=0}$ για κάθε κόμβο i του συνόλου A και $\mathbf{v_j=min\{c_{ij}\}}$ για κάθε κόμβο j του συνόλου B. Στη συνέχεια βρίσκει μία λύση M του αρχικού ακέραιου προβλήματος ανάθεσης έτσι ώστε να περιλαμβάνει μόνο αναθέσεις μεταξύ κόμβων i,j για τις οποίες ισχύει ότι $w_{ij} = 0$. Συγκεκριμένα, χρησιμοποιεί τον αλγόριθμο εύρεσης matching μεγίστου μήκους, ο οποίος είναι γνωστό ότι απαιτεί πολυωνυμική πολυπλοκότητα για να εκτελεστεί. Αν η M δεν αφήνει κανέναν κόμβο ακάλυπτο (unmatched) τότε η λύση M είναι βέλτιστη.

Αλλιώς, πρέπει να ανανεώσουμε την εφικτή λύση του dual προβλήματος μας ως εξής:

Έστω L είναι το σύνολο των κορυφών που μπορούν να προσεγγιστούν μέσα από ένα απευθείας κατευθυνόμενο μονοπάτι από μία μη ταιριασμένη (unmatched) κορυφή του συνόλου A. Τότε μπορεί να δειχτεί ότι δεν υπάρχει ακμή μεταξύ μιας κορυφής i του **A ∩ L** και μιας κορυφής j του **B − L** ώστε $\mathbf{w_{ij} = 0}$.

έστω:

$$\delta = \min_{i \in (A \cap L), j \in (B - L)} w_{ij}$$

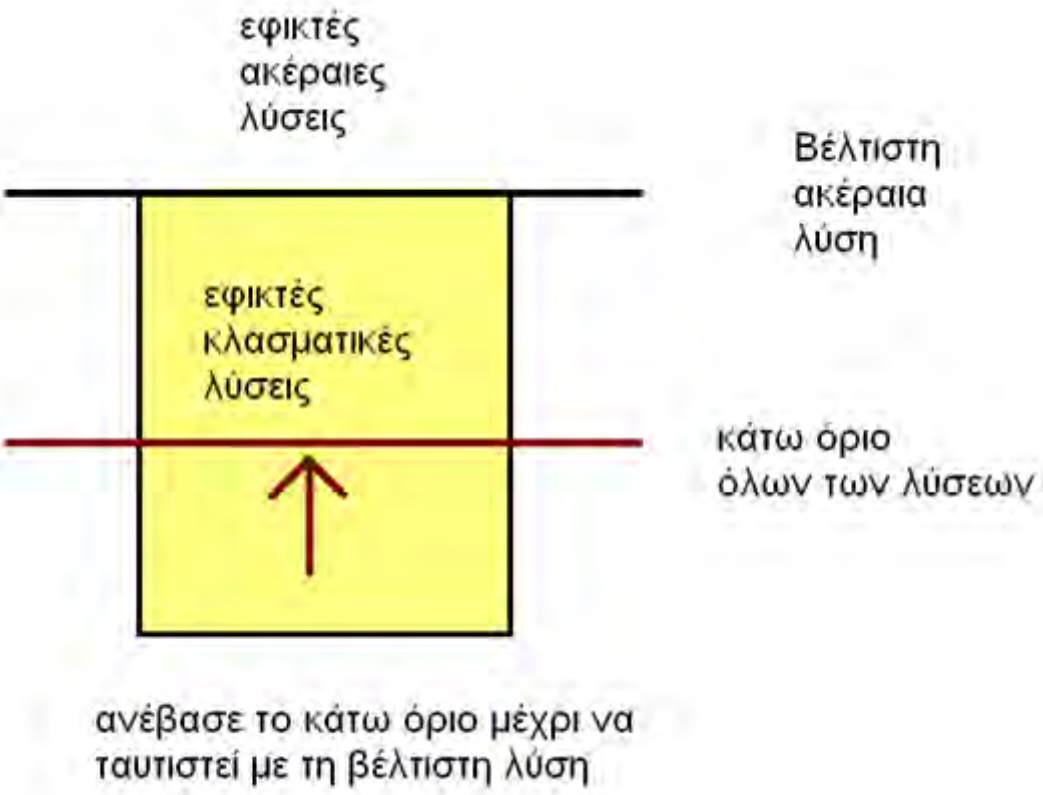
Από το προηγούμενο επιχείρημα ισχύει ότι **δ>0**. Τότε η λύση του dual προβλήματος ανανεώνεται ως εξής:

$$u_i = \begin{cases} u_i & i \in A - L \\ u_i + \delta & i \in A \cap L \end{cases}$$

$$v_j = \begin{cases} v_j & i \in B - L \\ v_j - \delta & j \in B \cap L \end{cases}$$

Μπορεί να δειχτεί ότι η νέα λύση του dual προβλήματος είναι αυστηρά μεγαλύτερη από πριν. Επομένως η τιμή του dual συνεχώς αυξάνεται και κάποια στιγμή είναι βέβαιο ότι η τιμή αυτή θα γίνει ίση με τη τιμή της λύσης του αντίστοιχου προβλήματος ακέραιης ανάθεσης.

Primal-Dual Αλγόριθμος - Απεικόνιση:



π

Περιγραφή του αλγορίθμου ως μια μέθοδος επεξεργασίας πινάκων

Ας δούμε τώρα πως μπορεί να εφαρμοστεί συστηματικά ο Ουγγρικός αλγόριθμος για τη λύση του προβλήματος ανάθεσης όταν χρησιμοποιούμε σημειογραφία πινάκων:

Η αλγοριθμική διαδικασία περιλαμβάνει **5 στάδια**:

Ζητείται η ανάθεση ελαχίστου κόστους των εργασιών a_1, a_2, a_3, a_4 στους εξυπηρετητές b_1, b_2, b_3, b_4 σύμφωνα με τον ακόλουθο πίνακα κόστους:

	b_1	**b_2**	**B_3**	**b_n**
a_1	1	8	4	1
a_2	5	7	6	5
a_3	3	5	4	2
a_n	3	1	6	3

Βήμα 1

Εύρεση των ελαχίστων τιμών των στηλών του πίνακα κόστους και αφαίρεση τους από τα στοιχεία κάθε στήλης.

	1	8	4	1
	5	7	6	5
	3	5	4	2
	3	1	6	3
min	1	1	4	1

Προκύπτει ο πίνακας:

0	7	0	0
4	6	2	4
2	4	0	1
2	0	2	2

Βήμα 2

Εύρεση του μέγιστου πλήθους ανεξάρτητων μηδενικών του πίνακα του προηγούμενου βήματος: Διαχωρισμός των μηδενικών του πίνακα σε «πλαισιούμενα» (ανεξάρτητα) και διαγραφόμενα.

Δύο ή περισσότερα μηδενικά στοιχεία του πίνακα χαρακτηρίζονται ως ανεξάρτητα όταν ποτέ δύο από αυτά δεν βρίσκονται στην ίδια γραμμή ή στήλη του πίνακα . Αν ο μέγιστος αριθμός ανεξάρτητων μηδενικών είναι n (n=4 στο παράδειγμα), τότε η βέλτιστη λύση αντιστοιχεί στις θέσεις αυτών και ο αλγόριθμος τερματίζεται. Αν ο μέγιστος αριθμός ανεξάρτητων μηδενικών είναι μικρότερος του n, ο αλγόριθμος συνεχίζεται στο επόμενο βήμα. Εδώ ο μέγιστος αριθμός ανεξάρτητων μηδενικών είναι 3 < n =4, οπότε ο αλγόριθμος συνεχίζεται στο βήμα 3.

0	7	~~0~~	~~0~~
4	6	2	4
2	4	**0**	1
2	**0**	2	2

Βήμα 3

Γίνονται οι ακόλουθες σημειώσεις:

1. Σημειώνονται οι γραμμές στις οποίες δεν υπάρχουν πλαισιούμενα μηδενικά
2. Σημειώνονται οι στήλες που έχουν διαγραφόμενο μηδενικό σε μια τουλάχιστον από τις σημειωθείσες γραμμές
3. Σημειώνονται οι γραμμές που έχουν πλαισιούμενο σε σημειωθείσα στήλη

Τα υποβήματα 2, 3 έχουν επαναληπτικό χαρακτήρα, οι δε σημειώσεις γίνονται όσο υπάρχουν αντίστοιχα μηδενικά.

Στο παράδειγμα, γίνεται σημείωση μόνο της δεύτερης γραμμής του πίνακα.

0	7	~~0~~	~~0~~	
4	6	2	4	*
2	4	**0**	1	
2	**0**	2	2	

Βήμα 4

Κάλυψη των μη σημειωμένων γραμμών και των σημειωμένων στηλών. Εύρεση της ελάχιστης τιμής των μη καλυμμένων στοιχείων του πίνακα.

	0	7	~~0~~	~~0~~	
	4	6	2	4	*
	2	4	**0**	1	
	2	**0**	2	2	

Min = 2

Βήμα 5

Αφαίρεση της ελάχιστης τιμής από τα μη καλυμμένα στοιχεία, πρόσθεσή της στα δύο φορές καλυμμένα στοιχεία (τομές καλυμμένων γραμμών - στηλών) και επάνοδος στο βήμα 2.

2η επανάληψη:

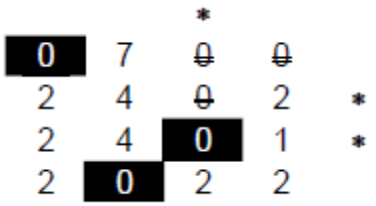
Βήμα 2

0	7	0	0
2	4	0	2
2	4	0	1
2	0	2	2

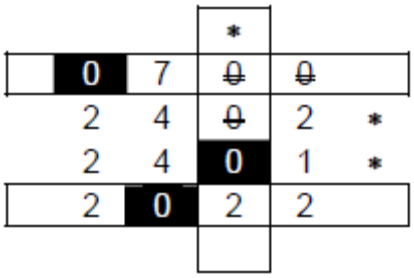
0	7	~~0~~	~~0~~
2	4	~~0~~	2
2	4	**0**	1
2	**0**	2	2

Μέγιστος αριθμός ανεξάρτητων μηδενικών είναι 3 < n =4, οπότε ο αλγόριθμος συνεχίζεται στο βήμα 3.

Βήμα 3



Βήμα 4



Min =1

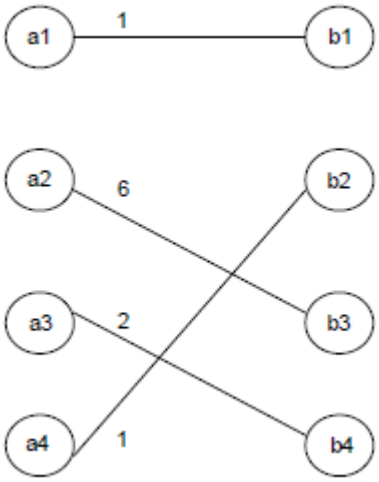
Βήμα 5

0	7	1	0
1	3	0	1
1	3	0	0
2	0	3	2

0	7	1	0
1	3	0	1
1	3	0	0
2	0	3	2

Επειδή ο αριθμός των ανεξάρτητων μηδενικών είναι 4, ο αλγόριθμος τερματίζεται με άριστη λύση την $x_{11}=1$, $x_{23}=1$, $x_{34}=1$, $x_{42}=1$ και κόστος $f = 1+1+6+2=10$ (οι μεταβλητές που δεν εμφανίζονται έχουν τιμή 0).

Η ανάθεση που αντιστοιχεί στη βέλτιστη λύση αποτυπώνεται στο παρακάτω γράφημα:



2.3 Παραλλαγές του προβλήματος ανάθεσης

Πέρα από το «κλασσικό» πρόβλημα ανάθεσης που περιγράψαμε, έχουν μελετηθεί διάφορες παραλλαγές του προβλήματος, τις πιο γνωστές από τις οποίες αναφέρουμε παρακάτω.

1) **Το αυστηρό πλήρες πρόβλημα ταιριάσματος**

Έστω ένας διμερής γράφος G=(V,E). Έστω E1,…,Εκ υποσύνολα του Ε και P1,…,Pκ θετικοί ακέραιοι. Το **αυστηρό πλήρες πρόβλημα ταιριάσματος** είναι να αποφασίσεις αν υπάρχει ένα πλήρες ταίριασμα M στο γράφο G το οποίο επιπλέον ικανοποιεί τους παρακάτω περιορισμούς:

|M∩Ej| <= Pj για κάθε j=1,…,k

Το πρόβλημα αυτό έ χει δειχτεί ότι ανήκει στην κατηγορία των NP-complete προβλημάτων (είναι μάλλον απίθανο να λυθεί βέλτιστα σε πολυωνυμικό χρόνο) με αναγωγή στο satisfiabihty problem of Boolean expressions. [10]

2) **b-ταίριασμα**

Έστω ότι κάθε κόμβος v του γράφου χαρακτηρίζεται από μία σταθερά b(v). Ένα **b-ταίριασμα** M του γράφου είναι ένα υποσύνολο των ακμών του έτσι ώστε, για κάθε κορυφή v, οι κορυφές που επιλέγονται στο ταίριασμα και ακουμπούν τη v να είναι το πολύ b(v) σε πλήθος.

Το πρόβλημα της εύρεσης ενός b-ταιριάσματος που περιέχει όσο γίνεται περισσότερες ακμές είναι επιλύσιμο σε πολυωνιμικό χρόνο ίσο με O((B^(1/2))*m), όπου m είναι το πλήθος των ακμών του γράφου, B είναι το άθροισμα των τιμών B(v) όλων των κορυφών [11].

3) **Ευσταθή προβλήματα ταιριάσματος**

Τα ευσταθή προβλήματα ανάθεσης αποτελούνται από ένα σύνολο πρακτόρων καθένας από τους οποίους εκδηλώνει μια ταξινομημένη λίστα με τις προτιμήσεις του σε σχέση με τους άλλους πράκτορες. Το πρόβλημα είναι να σχηματίσεις ένα ταίριασμα M των πρακτόρων έτσι ώστε να μην υπάρχουν 2 πράκτορες που να προτιμούν ο ένας τον άλλα σε σχέση με την ανάθεση M [12].

4) **ημι-ταιριάσματα**

Μια παραλλαγή του προβλήματος ανάθεσης είναι αυτή που πάλι αναζητάς την αντιστοίχηση μεταξύ πόρων και δραστηριοτήτων αλλά τώρα δεν απαιτούμε την 1-1 προς ένα αντιστοίχηση. Απαιτούμε μόνο ότι κάθε δραστηριότητα πρέπει να αντιστοιχηθεί σε έναν το πολύ πόρο όχι όμως και το αντίθετο. Αναζητούμε πάλι το ημι-ταίριασμα με το μικρότερο δυνατό συνολικό κόστος. Αυτή η παραλλαγή του προβλήματος είναι NP-complete [13].

2.4 Παρατηρήσεις

Όταν το πρόβλημα ανάθεσης είναι διατυπωμένο ως πρόβλημα μεγιστοποίησης (π.χ. μεγιστοποίηση το οφέλους από την ανάθεση ατόμων σε θέσεις εργασίας), επιλύεται με τον αλγόριθμο του Kuhn αφού πρώτα μετατραπεί σε πρόβλημα ελαχιστοποίησης. Τούτο επιτυγχάνεται σε ένα προκαταρκτικό στάδιο κατά το οποίο αφαιρείται το μέγιστο στοιχείο του πίνακα οφέλους από όλα τα στοιχεία του πίνακα. Στη συνέχεια ο αλγόριθμος εφαρμόζεται κανονικά στον νέο πίνακα.

Όταν ο αριθμός εργασιών είναι διαφορετικός από τον αριθμό εξυπηρετητών, όταν δηλαδή ο πίνακας του προβλήματος ανάθεσης δεν είναι τετραγωνικός, τούτος συμπληρώνεται με τον απαραίτητο αριθμό γραμμών ή στηλών και ο αλγόριθμος εφαρμόζεται κανονικά. Στην περίπτωση αυτή τα κόστη στις πρόσθετες γραμμές ή στήλες είναι μηδενικά.

Όταν μια ανάθεση δεν είναι δυνατό να πραγματοποιηθεί λόγω ενδεχομένων περιορισμών του προβλήματος, τίθεται για την ανάθεση αυτή μια πολύ μεγάλη τιμή κόστους προκειμένου να εξαιρεθεί από την βέλτιστη λύση.

3. Μια νέα απόπειρα επίλυσης του προβλήματος ανάθεσης

3.1 Άπληστος αλγόριθμος

Σε αυτήν την ενότητα προτείνουμε ένα νέο αλγόριθμο επίλυσης του προβλήματος ανάθεσης, τον λεγόμενο άπληστο αλγόριθμο. Ο αλγόριθμος αυτός είναι ευριστικός, δηλαδή βρίσκει μία λύση η οποία είναι πολύ κοντά στη βέλτιστη. Το πλεονέκτημα του είναι ότι απαιτεί λιγότερο χρόνο εκτέλεσης σε σχέση με τον Ουγγρικό αλγόριθμο.

Ο **άπληστος αλγόριθμος** λειτουργεί ως εξής:

Επαναληπτικά:

Επιλέγει σε κάθε βήμα την ακμή εκείνη με το μικρότερο βάρος, έστω την (i,j)

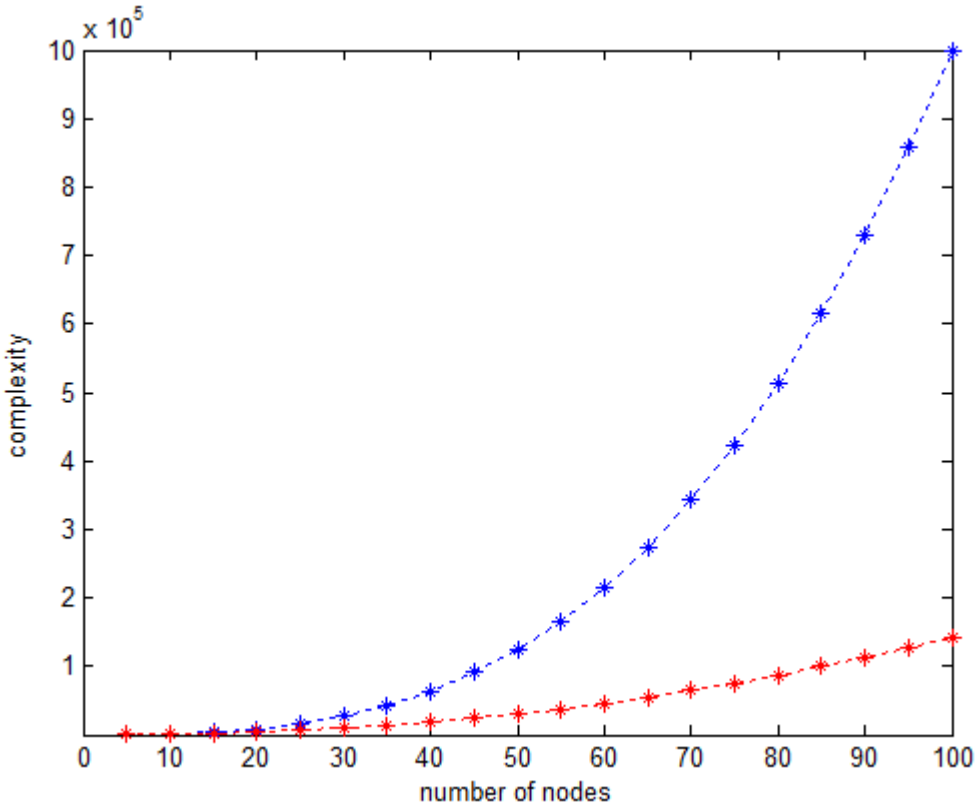
Εξάγει από το σύνολο των προς επιλογή ακμών τις ακμές εκείνες που ακουμπούν έστω και μία από τις κορυφές i,j.

Μέχρις ότου να επιλεγούν m ακμές.

3.2 Διαφορές με τον Ουγγρικό Αλγόριθμο

Είναι σαφές ότι η λύση που βρίσκει ο άπληστος αλγόριθμος είναι μη βέλτιστη. Παρόλα αυτά η πολυπλοκότητα που απαιτεί για να εκτελεστεί είναι μικρότερη από m^3 και συγκεκριμένα ίση με $O(2*m^2*\log m)$, η οποία δικαιολογείται ως εξής. Απαιτείται $2*m^2*\log m$ χρόνος για την ταξινόμηση των m^2 σε πλήθος βαρών των ακμών του αρχικού προβλήματος, χρησιμοποιώντας τον αλγόριθμο quicksort. Επιπλέον απαιτείται m*(m-1) χρόνος για την έξοδο m-1 ακμών από το χώρο των προς επιλογή ακμών σε κάθε ένα από τα m βήματα του αλγορίθμου.

Η επόμενη γραφική παράσταση συγκρίνει την χρονική πολυπλοκότητα εκτέλεσης των δύο αλγορίθμων για αυξανόμενες τιμές του μεγέθους εισόδου του προβλήματος ανάθεσης. Με μπλε χρώμα αναφέρεται ο Ουγγρικός αλγόριθμος και με κόκκινο ο άπληστος. Παρατηρούμε την σημαντική απόσταση των δύο πολυπλοκοτήτων ιδιαίτερα για μεγάλες τιμές εισόδου.



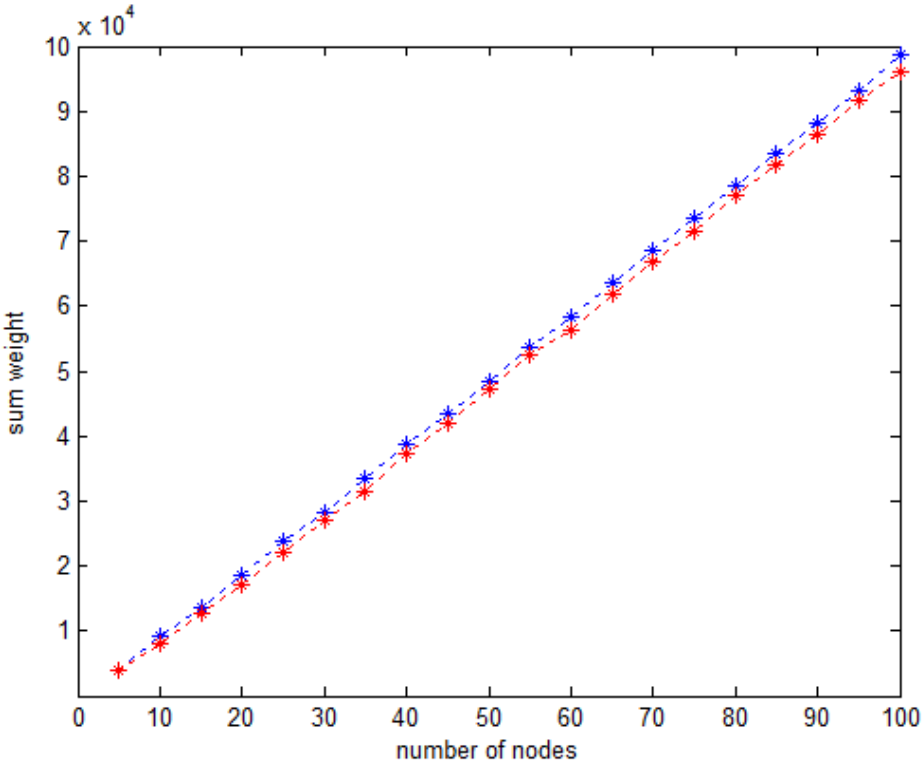
4. Αποτελέσματα προσομοιώσεων

Στην παρούσα ενότητα παρουσιάζουμε τα αποτελέσματα από τα πειράματα που εκτελέσαμε με στόχο την σύγκριση της απόδοσης των δύο αλγορίθμων (του Ουγγρικού και του άπληστου) σε πραγματικά στιγμιότυπα του προβλήματος ανάθεσης. Στόχος είναι να εξάγουμε χρήσιμα συμπεράσματα για το πώς μεταβάλλεται η απόσταση των δύο αλγορίθμων μεταβάλλοντας συγκεκριμένες τιμές των παραμέτρων εισόδου του προβλήματος ανάθεσης.

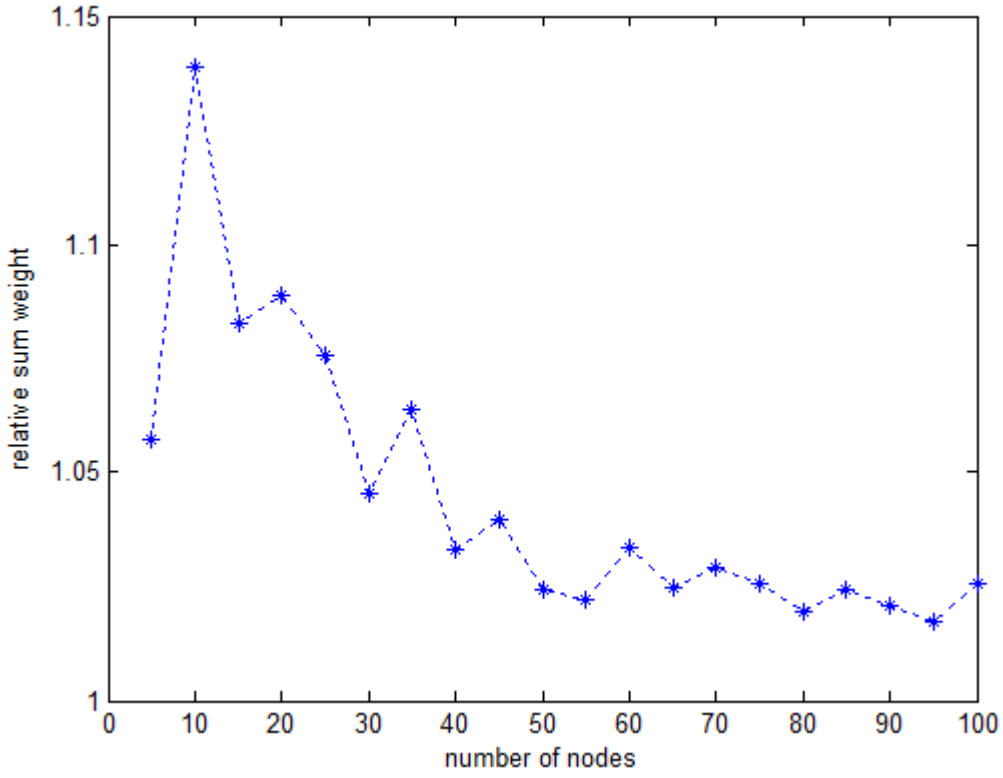
Σε όλα τα πειράματα από εδώ και κάτω αναφερόμαστε σε προβλήματα ανάθεσης που έχουν στόχο να μεγιστοποιήσουν το συνολικό κέρδος της ανάθεσης και όχι να ελαχιστοποιήσουν το συνολικό κόστος της ανάθεσης.

4.1 Μεταβολή μεγέθους στιγμιότυπων εισόδου του προβλήματος

Η επόμενη γραφική παράσταση παρουσιάζει την απόδοση των δύο αλγορίθμων για αυξανόμενες τιμές του μεγέθους εισόδου του προβλήματος ανάθεσης. Στα εν λόγω πειράματα αναφερόμαστε στο πρόβλημα της **μεγιστοποίησης της συνολικής ωφέλειας της ανάθεσης** και όχι στο ισοδύναμο πρόβλημα της ελαχιστοποίησης του συνολικού κόστους της ανάθεσης. Τα βάρη των ακμών του προβλήματος επιλέχτηκαν τυχαία με βάση την ομοιόμορφη κατανομή πιθανότητας από 0 ως 1000. Με μπλε χρώμα αναφέρεται ο Ουγγρικός αλγόριθμος και με κόκκινο ο άπληστος. Παρατηρούμε την μικρή απόσταση των δύο λύσεων.



Για να μελετήσουμε καλύτερα την διαφορά των δύο λύσεων απεικονίζουμε στην επόμενη γραφική παράσταση το λόγο των τιμών των δύο λύσεων σε καθεμία περίπτωση του προηγούμενου πειράματος. Παρατηρούμε ότι, με κάποιες διακυμάνσεις, ο λόγος αυτός συνεχώς μειώνεται, γεγονός που μας κάνει να συμπεράνουμε ότι όσο μεγαλώνουν τα στιγμιότυπα εισόδου του προβλήματος ανάθεσης τόσο λιγότερο χάνουμε σε απόδοση επιλέγοντας τον άπληστο αλγόριθμο έναντι του ουγγρικού. Άρα συμπεραίνουμε ότι η ποιότητα της λύσης που πετυχαίνει ο άπληστος αλγόριθμος βελτιώνεται όσο αυξάνεται το μέγεθος της εισόδου του προβλήματος.

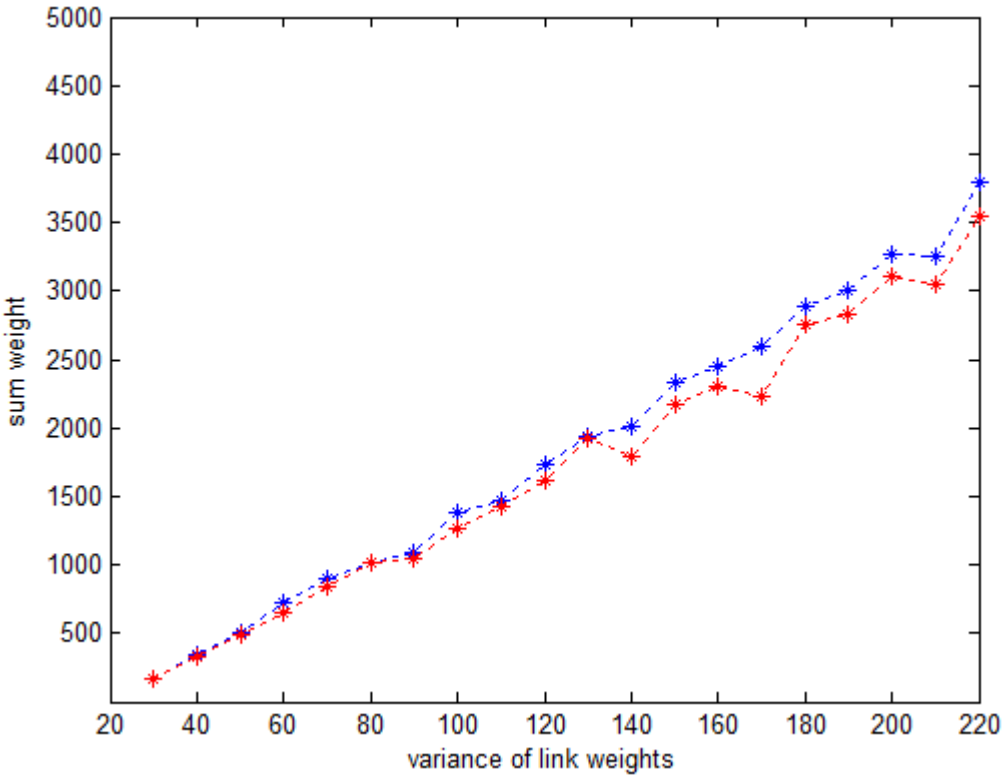


4.2 Μεταβολή κατανομής πιθανότητας των βαρών των ακμών του προβλήματος

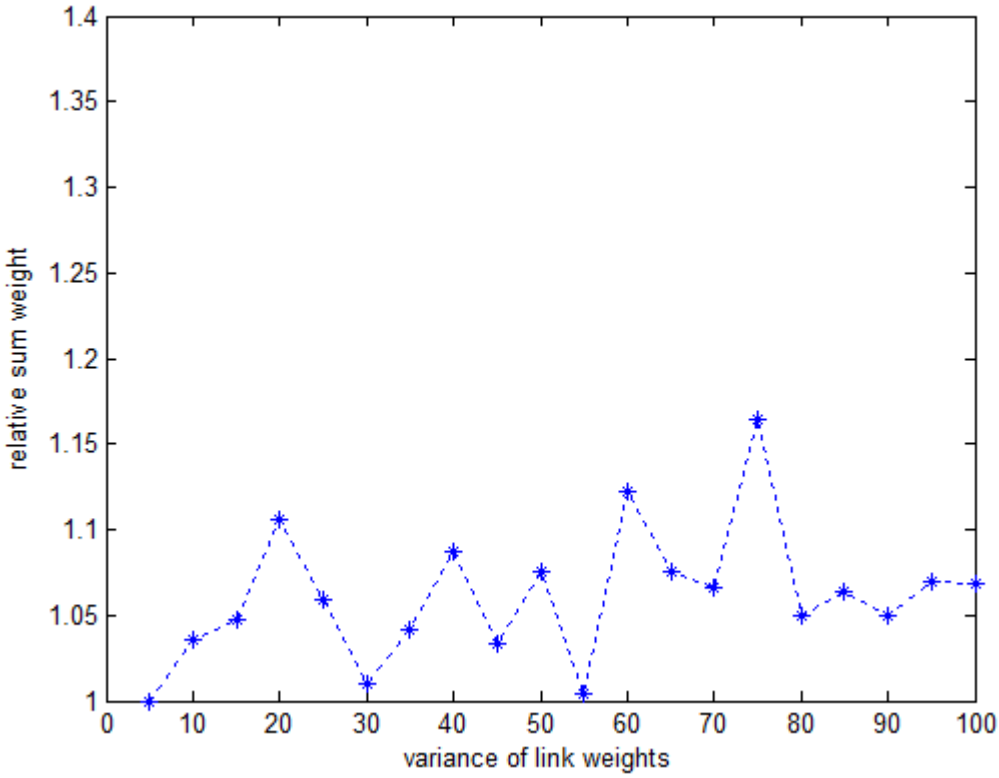
Σε αυτήν την υποενότητα εξετάζουμε κατά πόσο επηρεάζεται η απόσταση των λύσεων ανάμεσα στον Ουγγρικό και τον άπληστο αλγόριθμο ως συνάρτηση της κατανομής πιθανότητας των βαρών των ακμών του προβλήματος εισόδου. Θεωρούμε ότι το μέγεθος της εισόδου είναι σταθερό και ίσο με m=10.

Η επόμενη γραφική παράσταση παρουσιάζει την απόδοση των δύο αλγορίθμων για αυξανόμενες τιμές του εύρους επιλογής της ομοιόμορφης κατανομής των βαρών των ακμών του προβλήματος ανάθεσης. Στα εν λόγω πειράματα αναφερόμαστε στο πρόβλημα της **μεγιστοποίησης της συνολικής ωφέλειας της ανάθεσης** και όχι στο ισοδύναμο πρόβλημα της ελαχιστοποίησης του συνολικού κόστους της ανάθεσης. Η αύξηση του εύρους επιλογής της ομοιόμορφης κατανομής είναι ισοδύναμη με την αύξηση της διακύμανσης της εν λόγω κατανομής πιθανότητας.

Με μπλε χρώμα αναφέρεται ο Ουγγρικός αλγόριθμος και με κόκκινο ο άπληστος. Παρατηρούμε ότι η απόσταση μεγαλώνει κατά απόλυτη τιμή όσο αυξάνεται το εύρος επιλογής των βαρών των ακμών του γράφου εισόδου.



Για να μελετήσουμε καλύτερα την διαφορά των δύο λύσεων απεικονίζουμε στην επόμενη γραφική παράσταση το λόγο των τιμών των δύο λύσεων σε καθεμία περίπτωση του προηγούμενου πειράματος. Παρατηρούμε ότι, με κάποιες διακυμάνσεις, ο λόγος αυτός μένει σταθερός, γεγονός που μας κάνει να συμπεράνουμε ότι η ποιότητα της λύσης που πετυχαίνει ο άπληστος αλγόριθμος δεν επηρεάζεται από την διακύμανση της κατανομής πιθανότητας των βαρών των ακμών του προβλήματος.



5. *Επίλογος*

Στην παρούσα διπλωματική εργασία εξετάσαμε το πρόβλημα της ανάθεσης. Παρουσιάσαμε τον Ουγγρικό αλγόριθμο που χρησιμοποιείται για τη λύση του προβλήματος. Επιπλέον, προτείναμε ένα νέο ευριστικό αλγόριθμο, τον άπληστο αλγόριθμο, για την επίλυση του προβλήματος σε μικρότερο χρόνο. Τέλος, εκτελέσαμε πειράματα για την σύγκριση της απόδοσης των δύο αλγορίθμων σε περιοχές ενδιαφέροντος.

Ως αντικείμενο μελλοντικής εργασίας προτείνουμε την ενδελεχή σύγκριση των δύο αλγορίθμων, σε περιπτώσεις που η κατανομή πιθανότητας των βαρών των ακμών του προβλήματος εισόδου είναι διαφορετική από την ομοιόμορφη.

Παράρτημα

Στην ενότητα αυτή παραθέτουμε τον κώδικα που δημιουργήθηκε για την υλοποίηση του ουγγρικού και του άπληστου αλγορίθμου καθώς και των πειραμάτων που παρουσιάσαμε.
Η υλοποίηση έγινε στη γλώσσα προγραμματισμού C++.

Παρακάτω αναφέρουμε τις βιβλιοθήκες που χρειάζονται ώστε να εκτελεστεί ο εν λόγω κώδικας

```
#include<stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <emmintrin.h>
#include <time.h>
#include <sys/timeb.h>
#include <iostream>
#include <algorithm>
```

Στη συνέχεια αναφέρουμε τις καθολικές μεταβλητές του εν λόγω προγράμματος.

```
#define N 800                  //max number of vertices in one part
#define INF 100000000          //just infinity

float cost[N][N];              //cost matrix
int n, max_match;              //n workers and n jobs
int lx[N], ly[N];              //labels of X and Y parts
int xy[N];                     //xy[x] - vertex that is matched with x,
int yx[N];                     //yx[y] - vertex that is matched with y
bool S[N], T[N];               //sets S and T in algorithm
int slack[N];                  //as in the algorithm description
int slackx[N];                 //slackx[y] such a vertex, that
                               // l(slackx[y]) + l(y) - w(slackx[y],y) = slack[y]
int prev[N];                   //array for memorizing alternating paths
```

Παρακάτω παρουσιάζεται η βοηθητική συνάρτηση με όνομα init_labels που καλεί ο ουγγρικός αλγόριθμος.

```
void init_labels()
{
   memset(lx, 0, sizeof(lx));
   memset(ly, 0, sizeof(ly));
   for (int x = 0; x < n; x++)
                for (int y = 0; y < n; y++){
                        if(cost[x][y]>lx[x]){
                                lx[x] = cost[x][y];
                        }
                }

}
```

Παρακάτω παρουσιάζεται η βοηθητική συνάρτηση με όνομα update_labels που καλεί ο ουγγρικός αλγόριθμος.

```
void update_labels()
{
   int x, y, delta = INF;           //init delta as infinity
   for (y = 0; y < n; y++)          //calculate delta using slack
                if (!T[y]){
                        if(delta>slack[y])
        delta = slack[y];
                }
   for (x = 0; x < n; x++)          //update X labels
      if (S[x]) lx[x] -= delta;
   for (y = 0; y < n; y++)          //update Y labels
      if (T[y]) ly[y] += delta;
   for (y = 0; y < n; y++)          //update slack array
      if (!T[y])
         slack[y] -= delta;
}
```

Παρακάτω παρουσιάζεται η βοηθητική συνάρτηση με όνομα add_to_tree που καλεί ο ουγγρικός αλγόριθμος.

```
void add_to_tree(int x, int prevx)
//x - current vertex,prevx - vertex from X before x in the alternating path,
//so we add edges (prevx, xy[x]), (xy[x], x)
{
   S[x] = true;                    //add x to S
   prev[x] = prevx;                //we need this when augmenting
   for (int y = 0; y < n; y++)     //update slacks, because we add new vertex to S
      if (lx[x] + ly[y] - cost[x][y] < slack[y])
      {
         slack[y] = lx[x] + ly[y] - cost[x][y];
         slackx[y] = x;
      }
}
```

Παρακάτω παρουσιάζεται η βοηθητική συνάρτηση με όνομα augment που καλεί ο ουγγρικός αλγόριθμος.

```
void augment()                          //main function of the algorithm
{
   if (max_match == n) return;          //check wether matching is already perfect
   int x, y, root;                      //just counters and root vertex
   int q[N], wr = 0, rd = 0;            //q - queue for bfs, wr,rd - write and read
                                        //pos in queue
   memset(S, false, sizeof(S));         //init set S
   memset(T, false, sizeof(T));         //init set T
   memset(prev, -1, sizeof(prev));      //init set prev - for the alternating tree


   for (x = 0; x < n; x++)              //finding root of the tree
      if (xy[x] == -1)
      {
         q[wr++] = root = x;
         prev[x] = -2;
         S[x] = true;
         break;
      }


   for (y = 0; y < n; y++)              //initializing slack array
   {
      slack[y] = lx[root] + ly[y] - cost[root][y];
      slackx[y] = root;
   }


   while (true)                                 //main cycle
   {
      while (rd < wr)                           //building tree with bfs cycle
      {
         x = q[rd++];                           //current vertex from X part

         for (y = 0; y < n; y++)                //iterate through all edges in equality graph

            if (cost[x][y] == lx[x] + ly[y] &&  !T[y])
            {
               if (yx[y] == -1) break;          //an exposed vertex in Y found, so
                                                //augmenting path exists!
               T[y] = true;                     //else just add y to T
```

```
            q[wr++] = yx[y];                    //add vertex yx[y], which is matched
                                                //with y, to the queue
            add_to_tree(yx[y], x);              //add edges (x,y) and (y,yx[y]) to the tree
         }

      if (y < n) break;                         //augmenting path found!!!
   }
   if (y < n) break;                            //augmenting path found!!!


   update_labels();                             //augmenting path not found, so improve labeling
   wr = rd = 0;
   for (y = 0; y < n; y++)


   //in this cycle we add edges that were added to the equality graph as a
   //result of improving the labeling, we add edge (slackx[y], y) to the tree if
   //and only if !T[y] &&  slack[y] == 0, also with this edge we add another one
   //(y, yx[y]) or augment the matching, if y was exposed


      if (!T[y] &&  slack[y] == 0)
      {
         if (yx[y] == -1)                       //exposed vertex in Y found - augmenting path exists
         {
            x = slackx[y];
            break;
         }
         else
         {
            T[y] = true;                        //else just add y to T,
            if (!S[yx[y]])
            {
               q[wr++] = yx[y];                 //add vertex yx[y], which is matched with y, to the queue
               add_to_tree(yx[y], slackx[y]);   //and add edges (x,y) and (y,yx[y]) to the tree
            }
         }
      }

   if (y < n) break;                            //augmenting path found!
}
```

```
  if (y < n)                          //we found augmenting path!
  {
    max_match++;                      //increment matching
                                      //in this cycle we inverse edges along augmenting path


    for (int cx = x, cy = y, ty; cx != -2; cx = prev[cx], cy = ty)
    {
      ty = xy[cx];
      yx[cy] = cx;
      xy[cx] = cy;
    }
    augment();                                //recall function, go to step 1 of the algorithm
  }
```

Παρακάτω παρουσιάζεται η συνάρτηση με όνομα hungarian που υλοποιεί τον ουγγρικό αλγόριθμο.

```
float hungarian()
{
  float ret = 0;                      //weight of the optimal matching
  max_match = 0;                      //number of vertices in current matching
  memset(xy, -1, sizeof(xy));
  memset(yx, -1, sizeof(yx));
  init_labels();                      //step 0
  augment();                          //steps 1-3

      for (int x = 0; x < n; x++)  {  //forming answer there

        ret += cost[x][xy[x]];                //first is xy[x]...

      }


return ret;

}
```

Παρακάτω παρουσιάζεται η συνάρτηση με όνομα greedy που υλοποιεί τον άπληστο αλγόριθμο.

```
float greedy(){
        float f, max;
        int i,j,k, pos1, pos2;
        f=0;

        for(i=0; i<n; i++){                                  //a/a of chosen links
                max=0;
                pos1=-1;
                pos2=-1;
                                                             //find max weight link
                for(j=0; j<n; j++){                          //first part
                        for(k=0; k<n; k++){                  //second part
                                if(cost[j][k]>max){
                                        max=cost[j][k];
                                        pos1=j;
                                        pos2=k;
                                }
                        }
                }

                f=f+max;
                                                             //delete touched links
                for(j=0; j<n; j++){
                        cost[pos1][j]=0;
                        cost[j][pos2]=0;
                }
        }

        return f;
}
```

Παρακάτω παρουσιάζεται η κύρια συνάρτηση του προγράμματος:

```
int main(int argc, char *argv[]){

        n=10;                                   //size of graph
        int max=1000;                           //upper bound on value of weights
        int i,j;                                //local variables for loop
        float sum[20], sum1[20], sum2[20];


        for(max=20; max<=400; max=max+20)
       {
                                                //fill in weights of links

                for(i=0; i<n; i++){
                        for(j=0; j<n; j++){
                                cost[i][j]=(int)(rand()%max)+1;
                                //printf("%d\n",(int)cost[i][j]);
                        }
                }

                                                //call hungarian
                float c=hungarian();
                                                //call greedy
                float c2=greedy();

                for(i=0; i<n; i++){
                      //printf("node %d is matched with node %d\n",i,xy[i]);
                }



                sum[(max/20)-1]=c/c2;

                sum1[(max/20)-1]=c;

                sum2[(max/20)-1]=c2;

                printf("\nmax weight value: %f, %f. fraction=%f\n",c,c2,sum[(max/20)-1]);

        }
```

```
        FILE *graph = fopen ( "graph.m" , "w" );
        fprintf(graph, "function graph\n");

        fprintf(graph, "sum1=[\n");

        for (i = 0; i < 20 ; i++){
                        fprintf(graph, "%f ",sum1[i]);
        }

        fprintf(graph, "];\n");

        fprintf(graph, "sum2=[\n");

        for (i = 0; i < 20 ; i++){
                        fprintf(graph, "%f ",sum2[i]);
        }

        fprintf(graph, "];\n");
        fprintf(graph, "figure;

        \nplot([1:1:%d],sum1,':*b');
        \nhold on;\nylim([1 5000]);\nset(gca,'XTickLabel',[20:20:400])
        \nxlabel('variance of link weights');\nylabel('sum weight');\n",20);

        fprintf(graph, "hold on;

        \nplot([1:1:%d],sum2,':*r');\nhold on;\nylim([1 5000]);
        \nset(gca,'XTickLabel',[20:20:400])\nxlabel('variance of link weights');
        \nylabel('sum weight');\n",20);

        fprintf(graph, "sumweight=[\n");

        for (i = 0; i < 20 ; i++){
                        fprintf(graph, "%f ",sum[i]);
        }

        fprintf(graph, "];\n");

        fprintf(graph, "figure;\nplot([1:1:%d],sumweight,':*b');\nhold on;\nylim([1 1.4]);
        \nset(gca,'XTickLabel',[0:10:100])\nxlabel('variance of link weights');
        \nylabel('relative sum weight');\n",20);

        return 0;
}
```

Το εν λόγω πρόγραμμα καλεί επαναληπτικά τους δύο αλγορίθμους (ουγγρικό και άπληστο) και αποθηκεύει στους πίνακες sum1[], sum2[] και sum[] τις τιμές κόστους που πετυχαίνει ο ουγγρικός, ο άπληστος αλγόριθμος και τον λόγω αυτών των δύο αντίστοιχα.

Στη συνέχεια, δημιουργεί ένα matlab αρχείο, στο οποίο αποθηκεύει τον matlab κώδικα που αντιστοιχεί στη δημιουργία των γραφικών παραστάσεων που περιγράψαμε, περνώντας τις τιμές των sum1[], sum2[] και sum[].

Σύντομα βιογραφικά στοιχεία

Στο σημείο αυτό παραθέτουμε σύντομα βιογραφικά στοιχεία του δημιουργού του ουγγρικού αλγοριθμού, Kuhn.

Ο Χάρολντ William Kuhn (γεν. 1925) είναι ένας Αμερικανός μαθηματικός, που σπούδασε θεωρία παιγνίων. Κέρδισε το 1980 το John von Neumann Βραβείο θεωρίας μαζί με τον David Gale και Αλβέρτο W. Tucker. Ήταν καθηγητής των Μαθηματικών στο Πανεπιστήμιο του Πρίνστον, είναι γνωστός για τις Karush-Kuhn-Tucker συνθήκες, για την ανάπτυξη Kuhn πόκερ καθώς και η περιγραφή της ουγγρικής μεθόδου.

Είναι γνωστός για τη συνεργασία του με τον John Forbes Nash, ως υπότροφος μεταπτυχιακός φοιτητής, δια βίου φίλος και συνάδελφος. Ο Kuhn πιστώνεται ως ο σύμβουλος μαθηματικών στην ταινία της ζωής του Nash, A Beautiful Mind.

Μεγαλύτερος γιος του είναι ιστορικός Clifford Kuhn, γνωστός για την υποτροφία του για τον αμερικανικό Νότο και για τη συλλογή προφορική ιστορία. Ένας άλλος γιος, ο Nick Kuhn, είναι καθηγητής των μαθηματικών στο Πανεπιστήμιο της Βιρτζίνια. Ο μικρότερος γιος του, ο Jonathan Kuhn, είναι διευθυντής της Τέχνης και Αρχαιοτήτων για την Νέα Υόρκη Τμήμα Πάρκων & Αναψυχής.

Πηγή: http://en.wikipedia.org/wiki/Harold_W._Kuhn

Βιβλιογραφία

Ελληνική

[1] Γ. Σίσκος, Γραμμικός Προγραμματισμός, Εκδ. Νέων Τεχνολογιών, Αθήνα 1998

[2] Γραμμικός Προγραμματισμός, Δημήτρης Δεσπότης, Τμήμα Πληροφορικής - Εργαστήριο Συστημάτων Υποστήριξης Αποφάσεων, Πειραιάς 2007.

Ξένη

[1] J. Munkres, "Algorithms for the Assignment and Transportation Problems", Journal of the Society for Industrial and Applied Mathematics, 1957 March

[2] Kuhn H. W., "The Hungarian method for the assignment problem", Naval Research Logistics Quarterly, 1955

[3] R. A. Pilgrim, Munkres' Assignment Algorithm. Modified for Rectangular Matrices, Course notes, Murray State University

[4] Michel X. Goemans, Massachusetts Institute of Technology Handout 3 18.433: Combinatorial Optimization February 8th, 2007

[5] Mordecai J. Golin, Bipartite Matching and the Hungarian Method, Course Notes, Hong Kong University of Science and Technology

[6] Mike Dawes, The Optimal Assignment Problem, Course notes, University of Western Ontario.

[7] András Frank, Egervary Research Group, Pazmany P. setany 1/C, H1117, Budapest, Hungary, On Kuhn's Hungarian Method – A tribute from Hungary

[8] Kyiv National Taras Shevchenko University, community.topcoder.com/tc?module =Static&d1=tutorials&d2=hungarianAlgorithm

[9] Lecture: Fundamentals of Operations Research - Assignment Problem - Hungarian Algorithm, Prof. G. Srinivasan, Department of Management Studies, IIT Madras.

[10] Some Matching Problems for Bipartite Graphs, ALON ITAI, MICHAEL RODEH, STEVEN L. TANIMOTO

[11] H. Gabow. An efficient reduction technique for degree-constrained sub-graph and bidirected network flow problems. Journal of the ACM, pages 448-456, 1983.

[12] D. Guseld and R.W. Irving. The Stable Marriage Problem: Structure and Algorithms. MIT Press, 1989

[13] Assigning Sensors to Missions with Demands. A. Bar-Noy, T. Brown, M. Johnson, T. La Porta, O. Liu, H. Rowaih