EFFECTIVE PROGRAMMING, COMPUTATIONAL STUDY AND INTERNET VISUALIZATION OF NETWORK PROGRAMMING PROBLEMS ALGORITHMS

By
Charalampos Papamanthou

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
BACHELOR OF SCIENCE
AT
UNIVERSITY OF MACEDONIA
THESSALONIKI, GREECE
JUNE 2003

Αποτελεσματικός Προγραμματισμός, Υπολογιστική Μελέτη και Παρουσίαση στο Διαδίκτυο Αλγορίθμων Προβλημάτων Δικτυακού Προγραμματισμού

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ ΤΟΥ ΦΟΙΤΗΤΗ

Χαράλαμπου Ιωάννη Παπαμάνθου

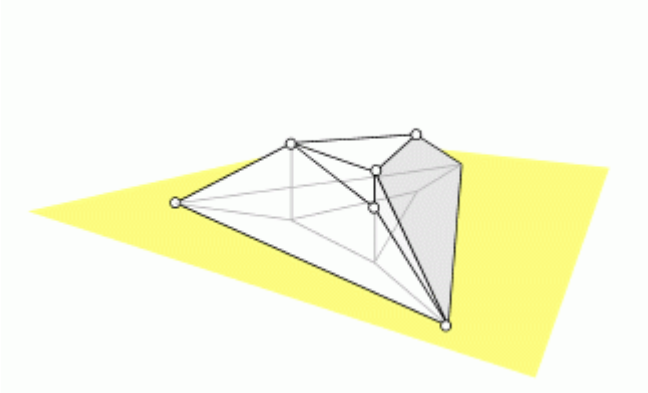
Εξεταστική Επιτροπή

Κωνσταντίνος Παπαρρίζος – Επιβλέπων Καθηγητής
Καθηγητής Τμήματος Εφαρμοσμένης Πληροφορικής Πανεπιστημίου Μακεδονίας

Νικόλαος Σαμαράς
Λέκτορας Τμήματος Εφαρμοσμένης Πληροφορικής Πανεπιστημίου Μακεδονίας

Ιωάννης Ρεφανίδης
Λέκτορας Τμήματος Εφαρμοσμένης Πληροφορικής Πανεπιστημίου Μακεδονίας

ΘΕΣΣΑΛΟΝΙΚΗ ΙΟΥΝΙΟΣ 2003



“Computer Science is no more about computers than astronomy is about telescopes.”

E. W. Dijkstra

ΕΥΧΑΡΙΣΤΙΕΣ

Ιδιαίτερες ευχαριστίες θα ήθελα να εκφράσω στον καθηγητή του τμήματος και επιβλέποντα της διπλωματικής εργασίας κύριο Κωνσταντίνο Παπαρρίζο για την εκπόνηση του τόσο ενδιαφέροντος θέματος, την αμέριστη συμπαράσταση και άριστη επιστημονική καθοδήγηση καθώς και για την αποδοτικότατη συνεργασία που αναπτύξαμε καθ' όλη τη διάρκεια εκπόνησης της διπλωματικής εργασίας.

Θερμές ευχαριστίες θα ήθελα να εκφράσω στον λέκτορα του τμήματος και μέλος της τριμελούς εξεταστικής επιτροπής κύριο Νικόλαο Σαμαρά για την μεγάλη του συνεισφορά στην διεξαγωγή της υπολογιστικής μελέτης καθώς και για τις πολύ σημαντικές και εύστοχες παρατηρήσεις του σε θεωρητικά και προγραμματιστικά θέματα.

Επίσης, ευχαριστώ τον λέκτορα του τμήματος και μέλος της τριμελούς εξεταστικής επιτροπής κύριο Ιωάννη Ρεφανίδη για τα σχόλια και τις διορθώσεις του καθώς και τους υποψήφιους διδάκτορες Άγγελο Σιφαλέρα, Κωνσταντίνο Δόσιο και Παναγιώτη Καραγιάννη για την υποστήριξη και το ενδιαφέρον τους.

Τέλος, θα ήθελα να ευχαριστήσω το προσωπικό του κέντρου Υπολογιστών του Πανεπιστημίου Μακεδονίας για τη βοήθειά τους στο πρωταρχικό στάδιο διεκπεραίωσης της υπολογιστικής μελέτης.

Θεσσαλονίκη, Ιούνιος 2003

Χαράλαμπος Ι. Παπαμάνθου

it0067@uom.gr

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1

ΕΙΣΑΓΩΓΙΚΕΣ ΕΝΝΟΙΕΣ

1.1 ΕΙΣΑΓΩΓΗ

Στα πλαίσια των μαθημάτων Θεωρίας Αλγορίθμων, Δικτυακού και Μαθηματικού Προγραμματισμού που διδάσκονται στο Τμήμα Εφαρμοσμένης Πληροφορικής του Πανεπιστημίου Μακεδονίας, μου ανατέθηκε κατά το Ζ' εξάμηνο σπουδών από τον καθηγητή κύριο Κωνσταντίνο Παπαρρίζο η συγγραφή διπλωματικής εργασίας με θέμα ***«Αποτελεσματικός Προγραμματισμός, Υπολογιστική Μελέτη και Παρουσίαση στο Διαδίκτυο Αλγορίθμων Προβλημάτων Δικτυακού Προγραμματισμού»***.

Ο Δικτυακός Προγραμματισμός αποτελεί ένα επιστημονικό πεδίο που υπάγεται στις εφαρμογές του Γραμμικού Προγραματισμού. Ασχολείται με προβλήματα βελτιστοποίησης που βρίσκουν εφαρμογή σε δίκτυα ή κατευθυνόμενα γραφήματα. Τρία από τα σημαντικότερα προβλήματα που επιλύει ο Δικτυακός Προγραμματισμός είναι το Πρόβλημα Ροής Ελαχίστου Κόστους (*The Minimum Cost Network Flow Problem*), το Πρόβλημα Της Μεταφοράς (*The Transportation Problem*) και το Πρόβλημα της Αντιστοίχησης (*The Assignment Problem*). Οι αλγόριθμοι που αναπτύχθηκαν και αναπτύσσονται για την επίλυση αυτών των προβλημάτων οφείλουν την ραγδαία ανάπτυξη τους στην αυξανόμενη υπολογιστική ισχύ που προσφέρει η ανάπτυξη της Επιστήμης των Υπολογιστών και της Τεχνολογίας. Συνεπώς, εύκολα μπορεί να εξαχθεί το συμπέρασμα ότι ο Δικτυακός Προγραμματισμός αποτελεί ένα σημαντικό πεδίο έρευνας και εφαρμογής στο χώρο των Υπολογιστών και της Πληροφορικής.

Οι αλγόριθμοι επίλυσης αυτών των προβλημάτων συνδέονται μεταξύ τους σε μεγάλο βαθμό. Ειδικότερα, μπορούμε να πούμε ότι οι αλγόριθμοι που επιλύουν τα προβλήματα Αντιστοίχησης και Μεταφοράς αποτελούν εξειδίκευση των αλγορίθμων που επιλύουν το πρόβλημα ροής ελαχίστου κόστους. Η εξειδίκευση αυτή πραγματοποιείται για την επίτευξη καλύτερων υπολογιστικών αποδόσεων κατά τη διάρκεια επίλυσης των συγκεκριμένων προβλημάτων. Στη διπλωματική εργασία που ακολουθεί θα αναφερθούμε εκτενώς στα δύο τελευταία προβλήματα και θα παρουσιάσουμε τους διάφορους αλγορίθμους που έχουν αναπτυχθεί για την επίλυση τους. Παράλληλα, θα παρουσιασθεί και μία υπολογιστική μελέτη, τόσο θεωρητική, εφαρμόζοντας τη μαθηματική θεωρία Ανάλυσης Αλγορίθμων και υπολογίζοντας την πολυπλοκότητα μερικών αλγορίθμων, όσο και εμπειρική, που προκύπτει από την εκτέλεση των αλγορίθμων αυτών σε ένα υπολογιστικό σύστημα, έτσι ώστε να επιτευχθεί μια ανάλυση και σύγκριση της επίδοσης τους για διαφορετικά μεγέθη προβλήματος και διαφορετικά είδη δεδομένων.

Γενικά, τα προβλήματα με τα οποία ασχολείται ο Δικτυακός Προγραμματισμός είναι *NP* – hard προβλήματα. Δηλαδή ανήκουν σε μια κλάση προβλημάτων, των οποίων η επίλυση απαιτεί αυξανόμενη υπολογιστική ισχύ. Για αυξανόμενα μεγέθη προβλήματος, ο χρόνος επίλυσης παύει πλέον πρακτικά να εξαρτάται από τη τεχνολογία και την υπολογιστική ισχύ που διαθέτουμε και η παραγωγή μιας λύσης αποτελεί μια ιδιαίτερα χρονοβόρα διαδικασία .

Όσον αφορά το προγραμματιστικό μέρος της εργασίας θα πρέπει να επισημάνουμε τα εξής: Οι αλγόριθμοι που θα εξεταστούν αρχικά προγραμματίστηκαν σε Matlab 6, που αποτελεί ένα εξαιρετικά ευέλικτο εργαλείο δομημένου προγραμματισμού, ιδιαίτερα όταν προγραμματίζονται αλγόριθμοι υψηλής μαθηματικής δυσκολίας, όπως είναι οι αλγόριθμοι Δικτυακού Προγραμματισμού. Το Matlab παρέχει τη δυνατότητα στον προγραμματιστή να ελέγχει και να βλέπει τα περιεχόμενα μιας μεταβλητής πολύ γρήγορα, πετυχαίνοντας έτσι μεγάλη ευκολία στην επεξεργασία σύνθετων δομών δεδομένων καθώς και στην ανίχνευση λογικών λαθών **[SS]**,**[PK]**. Τα προγράμματα αυτά χρησιμοποιήθηκαν για τη διεξαγωγή της εμπειρικής υπολογιστικής μελέτης, αφού το Matlab παρέχει τη δυνατότητα διαχείρισης χρόνου καθώς και ευέλικτα εργαλεία για υπολογιστικές μελέτες και βελτιώσεις (*Profiler*). Στη συνέχεια, βασιζόμενοι στην ορθότητα των προγραμμάτων του Matlab,

προγραμματίσαμε τους αλγορίθμους στην αντικειμενοστραφή γλώσσα Java, εισάγοντας παράλληλα και το οπτικό μέρος των αλγορίθμων, με αποτέλεσμα να παράγουμε ολοκληρωμένες εφαρμογές που θα μπορούν να εξαχθούν στον παγκόσμιο ιστό ως μικροεφαρμογές (java applets) και να χρησιμοποιηθούν για τη διδασκαλία και την παρουσίαση αυτών των αλγορίθμων **[PP]**, **[PPS]**.

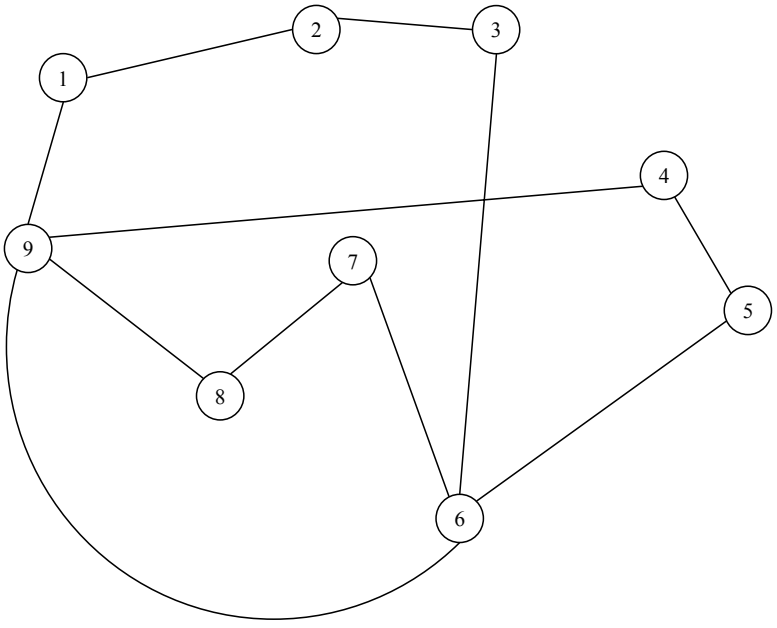
Παρακάτω, θα αναφερθούμε σε θεωρητικές έννοιες που είναι απαραίτητες για την κατανόηση των αλγορίθμων και στον ορισμό των προβλημάτων που θα παρουσιασθούν στην εργασία.

1.2 ΒΑΣΙΚΕΣ ΕΝΝΟΙΕΣ ΘΕΩΡΙΑΣ ΓΡΑΦΗΜΑΤΩΝ

1.2.1 Ορισμοί και Ιδιότητες Γραφημάτων

Είναι γεγονός, ότι πολλά από τα προβλήματα της καθημερινότητας μπορούν να παρασταθούν και να περιγραφούν με ένα σύνολο σημείων του επιπέδου, τα οποία συνδέονται μεταξύ τους με γραμμές. Παραδείγματος χάριν, ένα οδικό ή ένα σιδηροδρομικό δίκτυο μπορεί να παρασταθεί με αυτόν τον τρόπο, όσο πολύπλοκο και αν είναι αυτό. Ιδιαίτερα, στο χώρο της Επιστήμης των Υπολογιστών, αυτή η αναπαράσταση βρίσκει συχνά εφαρμογή σε πολλούς τομείς, όπως τα δίκτυα επικοινωνίας υπολογιστών. Όλα αυτά οδήγησαν στην ανάπτυξη μιας μαθηματικής θεωρίας που ονομάστηκε *«Θεωρία Γραφημάτων»* . Η Θεωρία Γραφημάτων αποτελεί μια αυστηρή μαθηματική θεωρία που γνώρισε μεγάλη ανάπτυξη με την πρόοδο της Πληροφορικής.

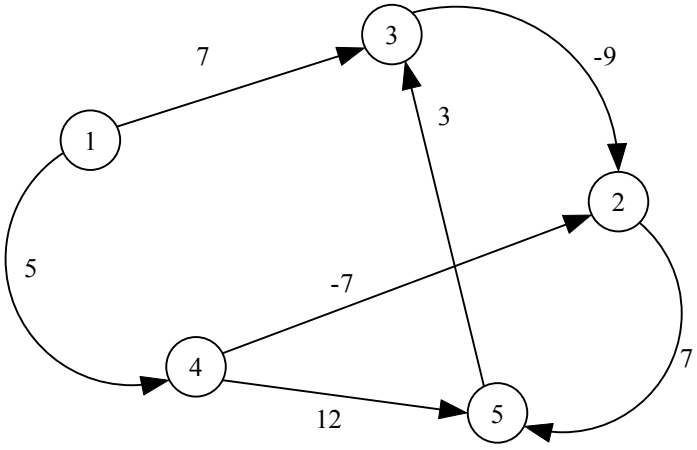
Το θεμελιώδες στοιχείο της Θεωρίας Γραφημάτων είναι το γράφημα. Το γράφημα ή γράφος (graph) είναι μια δομή που αποτελείται από ένα σύνολο κορυφών (vertices) ή κόμβων (nodes) και ένα σύνολο ακμών (edges). Γενικά, μπορούμε να πούμε ότι ένα γράφημα είναι μια διατεταγμένη δυάδα δύο συνόλων N, A,όπου το σύνολο N περιέχει όλες τις κορυφές του γραφήματος και το σύνολο A περιέχει στοιχεία της μορφής (i, j) όπου $i, j \in N$ τα οποία αντιπροσωπεύουν τους συνδέσμους μεταξύ των κορυφών του γραφήματος. Γενικά, συμβολίζουμε ένα γράφημα με $G = (N, A)$ και λέμε ότι αποτελείται από $|N|$ κορυφές και $|A|$ ακμές, όπου $|X|$ ο πληθικός αριθμός του συνόλου X .Στην εικόνα 1.0, βλέπουμε τη σχηματική αναπαράσταση ενός γραφήματος 9 κόμβων και 11 ακμών.



Εικόνα 1.0 - Ένα γράφημα 9 κόμβων

Εύκολα μπορούμε να καθορίσουμε τα σύνολα N και A που θα αντιστοιχούν στο παραπάνω γράφημα. Συνεπώς η μαθηματική αναπαράσταση του παραπάνω γραφήματος θα είναι $G=(N,A)$,όπου $N=\{1,2,3,4,5,6,7,8,9\}$ εκφράζει το σύνολο των κόμβων του γραφήματος και $A=\{(1,2),(1,9),(2,3),(3,6),(4,5),(4,9),(5,6),(6,7),(7,8),(8,9),(9,6)\}$ εκφράζει το σύνολο των ακμών του γραφήματος.

Τα γραφήματα χωρίζονται στα προσανατολισμένα και στα μη προσανατολισμένα γραφήματα. Εμείς θα ασχοληθούμε με τα προσανατολισμένα γραφήματα που αλλιώς ονομάζονται κατευθυνόμενα γραφήματα ή δίκτυα. Τα προσανατολισμένα γραφήματα είναι ειδικές περιπτώσεις των μη προσανατολισμένων γραφημάτων από τα οποία και προκύπτουν αν αντικαταστήσουμε κάθε ακμή (i,j) ενός μη προσανατολισμένου γραφήματος με δύο τόξα (i,j) και (j,i).Συνεπώς, συμπεραίνουμε ότι σε ένα προσανατολισμένο γράφημα $G=(N,A)$ ισχύει $(i,j)\neq(j,i)\ \ \forall(i,j)\in A$. Αν σε κάθε τόξο $(i,j)\in A$ και σε κάθε κόμβο $i\in N$ προσαρτήσουμε και έναν πραγματικό αριθμό c_{ij} και b_i αντίστοιχα τότε το κατευθυνόμενο γράφημα ονομάζεται δίκτυο(Εικόνα 1.1) και μπορεί να αποτελέσει ένα μοντέλο μελέτης πολλών πραγματικών προβλημάτων, όπως η ροή ηλεκτρικού φορτίου σε ένα κλειστό κύκλωμα ρεύματος.



Εικόνα 1.1 – Ένα δίκτυο 5 κόμβων

Στη συνέχεια θα προχωρήσουμε σε μερικούς ορισμούς που χρειάζονται στην κατανόηση των προβλημάτων που θα περιγράψουμε και των αλγορίθμων που θα χρησιμοποιήσουμε για την επίλυση των προβλημάτων. Θα λέμε ότι ένα τόξο (i,j) προσπίπτει στους κόμβους i και j και ότι οι κόμβοι i και j προσπίπτουν στο τόξο (i,j).Επίσης, ένα γράφημα ονομάζεται συνεκτικό(*connected*) αν για κάθε ζευγάρι κόμβων υπάρχει και μια διαδρομή(*path*) που να τους συνδέει. Επίσης, αν υπάρχει τουλάχιστον ένας απομονωμένος κόμβος, τότε ο γράφος ονομάζεται μη συνεκτικός(*unconnected*).

Στη συνέχεια θα δώσουμε τον ορισμό του βαθμού ενός κόμβου και του πλήρους γραφήματος. Βαθμός(*degree*) ενός κόμβου είναι ο αριθμός των τόξων που προσπίπτουν σε αυτόν. Ειδικότερα, ο αριθμός των τόξων που έχουν αρχή έναν κόμβο i ονομάζεται έξω βαθμός(*out degree*) του κόμβου i και συμβολίζεται με $d^-(i)$ και αντιστοίχως ο αριθμός των τόξων που έχουν τέλος τον κόμβο i ονομάζεται έσω βαθμός του κόμβου αυτού και συμβολίζεται με $d^+(i)$.Τέλος, πλήρες γράφημα ονομάζεται το γράφημα εκείνο που περιέχει όλες τις δυνατές ακμές που μπορούν να προκύψουν από τη σύνδεση των κορυφών του. Στη συνέχεια θα περιγράψουμε μεθόδους αποθήκευσης γραφημάτων.

1.2.2 Αποθήκευση Γραφημάτων και Δικτύων

Έστω έχουμε ένα μη κατευθυνόμενο γράφημα $G=(N,A)$. Στη συνέχεια θα περιγράψουμε πώς μπορούμε να αποθηκεύσουμε αυτό το γράφημα έτσι ώστε να μπορεί να επεξεργασθεί από τον ηλεκτρονικό υπολογιστή. Η πιο συνηθισμένη μέθοδος αποθήκευσης γραφημάτων είναι η χρησιμοποίηση πινάκων. Οι βασικότερες μέθοδοι αποθήκευσης χρησιμοποιώντας πίνακες είναι δύο:

α) Πίνακας Γειτνιάσεως(*Adjacency Matrix*)

Ο πίνακας γειτνιάσεως είναι ένας $|N|\times|N|$ πίνακας a, τα στοιχεία του οποίου παίρνουν τιμές από το σύνολο $\{0,1\}$ και ορίζονται ως εξής:

$$a(i,j)=\begin{cases}1, \forall (i,j)\in A\\ 0, \forall (i,j)\notin A\end{cases}$$

Δηλαδή ο πίνακας γειτνιάσεως είναι ένας πίνακας του οποίου τα στοιχεία του είναι μονάδες μόνο όταν η γραμμή και η στήλη όπου ανήκουν αποτελούν μια ακμή του γραφήματος, αλλιώς είναι μηδενικά. Συνεπώς, ο πίνακας γειτνίασης για το δίκτυο του σχήματος 1.1 θα είναι:

$$a=\begin{bmatrix}0&0&1&1&0\\0&0&0&0&1\\0&0&1&0&0\\0&1&0&0&1\\0&0&1&0&0\end{bmatrix}$$

Εύκολα μπορεί να αντιληφθεί κανείς ότι ο συγκεκριμένος τρόπος αποθήκευσης δεν είναι καθόλου οικονομικός. Αυτό συμβαίνει επειδή τις περισσότερες φορές τα μηδενικά των πινάκων αυτών είναι πάρα πολλά σε σχέση με τις μονάδες με αποτέλεσμα να χρησιμοποιείται χώρος αποθήκευσης της τάξεως $|N|^2$ όταν η σημαντική πληροφορία που πρέπει να αποθηκεύσουμε απαιτεί $|A|$ θέσεις. Συγκεκριμένα, όταν χρησιμοποιούμε πίνακα γειτνίασης για την αποθήκευση μη κατευθυνόμενων γραφημάτων, αυτός θα είναι συμμετρικός, με αποτέλεσμα τα στοιχεία κάτω από την κύρια διαγώνιο να μην χρειάζονται αποθήκευση.

Επίσης, πρέπει να επισημανθεί ότι ο πίνακας γειτνίασης μπορεί με την ίδια λογική να χρησιμοποιηθεί για την ταυτόχρονη αποθήκευση του γραφήματος – δικτύου και των βαρών που αντιστοιχούν σε κάθε ακμή. Στην περίπτωση αυτή, ο πίνακας που θα αναπαριστάνει το δίκτυο του σχήματος 1.1 θα είναι ο εξής:

$$a=\begin{bmatrix}\infty&\infty&7&5&\infty\\\infty&\infty&\infty&\infty&7\\\infty&\infty&-9&\infty&\infty\\\infty&-7&\infty&\infty&12\\\infty&\infty&3&\infty&\infty\end{bmatrix}$$

Ένας αριθμός δηλαδή σε μία θέση του πίνακα υποδηλώνει την ύπαρξη και το βάρος της αντίστοιχης ακμής του γραφήματος. Η δεύτερη μορφή του πίνακα προσπτώσεως χρησιμοποιείται πολύ πιο συχνά, αφού προσφέρει οικονομικότερο τρόπο αποθήκευσης των δεδομένων του προβλήματος.

β. Πίνακας Προσπτώσεως(*Indicence Matrix*)

Ο πίνακας προσπτώσεως είναι ένας $|N|\times|A|$ πίνακας a τα στοιχεία του οποίου ορίζονται διαφορετικά για κατευθυνόμενα και μη κατευθυνόμενα γραφήματα. Ειδικότερα, κάθε στήλη του πίνακα προσπτώσεως αντιστοιχεί σε μία ακμή ή ένα τόξο των γραφημάτων που θέλουμε να αποθηκεύσουμε. Συνεπώς, όταν θέλουμε να αποθηκεύσουμε ένα μη κατευθυνόμενο γράφημα, χρησιμοποιώντας ένα πίνακα προσπτώσεως a, τότε θα έχουμε:

$$a(i,j)=\begin{cases}1, & \text{αν η ακμη } j \text{ προσπιπτει στην κορυφη } i\\ 0, & \text{διαφορετικα}\end{cases}$$

Όταν όμως θέλουμε να αποθηκεύσουμε ένα κατευθυνόμενο γράφημα χρησιμοποιώντας τον πίνακα προσπτώσεως τότε θα έχουμε:

$$a(i,j)=\begin{cases}1 & , \text{ αν η κορυφη } i \text{ ειναι αρχη της ακμης } j\\ -1 & , \text{ αν η κορυφη } i \text{ ειναι τελος της ακμης } j\\ 0 & , \text{ διαφορετικα}\end{cases}$$

Λαμβάνοντας υπ' όψιν τα παραπάνω η αποθήκευση του δικτύου του σχήματος 1.2 μπορεί να επιτευχθεί χρησιμοποιώντας τον παρακάτω 5×7 πίνακα προσπτώσεως:

$$a=\begin{bmatrix}1 & 1 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 1 & -1 & -1 & 0 & 0\\ -1 & 0 & 0 & 1 & 0 & 0 & -1\\ 0 & -1 & 0 & 0 & 1 & 1 & 0\\ 0 & 0 & -1 & 0 & 0 & -1 & 1\end{bmatrix}$$

,όπου $t=\begin{bmatrix}(1,3) & (1,4) & (2,5) & (3,2) & (4,2) & (4,5) & (5,3)\end{bmatrix}$ είναι το διάνυσμα στο οποίο κρατάμε τη σειρά με την οποία αποθηκεύονται τα τόξα του γραφήματος.

1.2.3 Πυκνότητα Γραφήματος

Έστω έχουμε ένα γράφημα $G=(N,A)$. Μια πολύ σημαντική ιδιότητα των γραφημάτων είναι η πυκνότητα του γραφήματος. Πυκνότητα ενός γραφήματος ονομάζεται το πηλίκο του αριθμού των ακμών του γραφήματος προς τον αριθμό των ακμών του αντίστοιχου πλήρους γραφήματος.

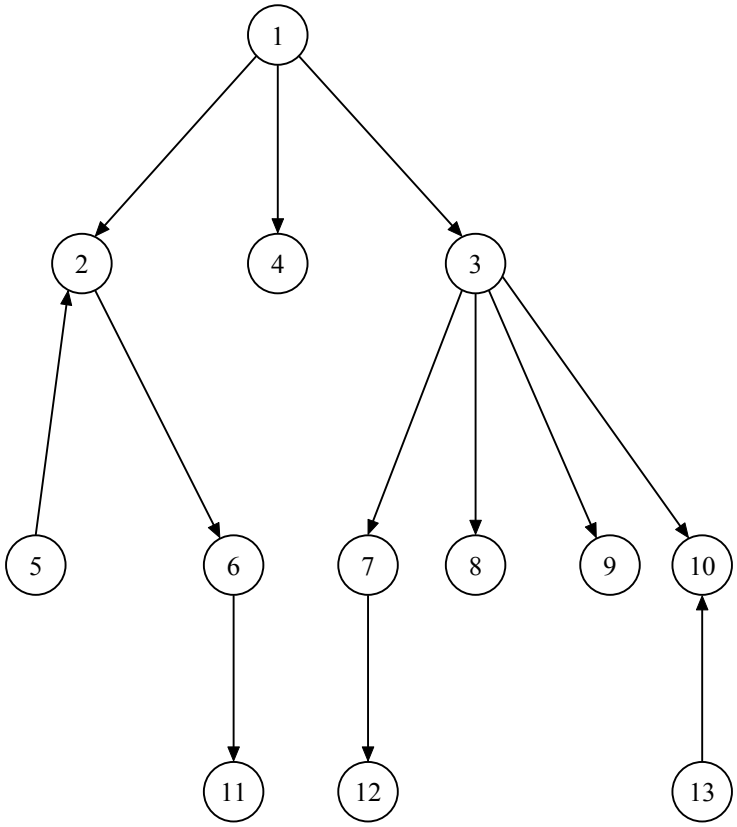
Αν το γράφημά μας είναι προσανατολισμένο και αποτελείται από $n=|N|$ κορυφές και $e=|A|$ ακμές, η πυκνότητά του θα ισούται με $d=\dfrac{2e}{n(n-1)}$, αφού $n(n-1)/2$ είναι το πλήθος των τόξων ενός πλήρους κατευθυνόμενου γραφήματος n κόμβων, ενώ όταν έχουμε ένα μη κατευθυνόμενο γράφημα θα έχουμε $d=\dfrac{e}{n(n-1)}=\dfrac{e}{n(n-1)}$, αφού $n(n-1)$ είναι το πλήθος των τόξων ενός πλήρους μη κατευθυνόμενου γραφήματος n κόμβων . Εύκολα μπορεί να καταλάβει κανείς ότι η πυκνότητα είναι ένας αριθμός μεταξύ του μηδενός και της μονάδας. Γραφήματα που έχουν πυκνότητα μεγάλη ονομάζονται πυκνά (*dense*), ενώ γραφήματα με μικρή πυκνότητα ονομάζονται αραιά (*sparse*).Η πυκνότητα είναι ένα πολύ σημαντικό μέγεθος ιδιαίτερα στις υπολογιστικές μελέτες που αφορούν αλγορίθμους θεωρίας Γραφημάτων και Δικτυακού Προγραμματισμού. Τα αποτελέσματα δηλαδή των

υπολογιστικών μελετών διαφέρουν ανάλογα με την πυκνότητα του γραφήματος που χρησιμοποιούμε και με αποτέλεσμα να βγάζουμε χρήσιμα συμπεράσματα .

1.2.4 Δέντρα

1.2.4.1 Γενικά

Τα δέντρα(*trees*) ανήκουν σε μια πολύ σημαντική κατηγορία γραφημάτων. Ένα γράφημα G με n κορυφές ονομάζεται δέντρο εάν και μόνον αν έχει $n-1$ ακμές και είναι συνεκτικό. Τα δέντρα θα είναι η σημαντικότερη δομή δεδομένων την οποία θα χρησιμοποιούν οι αλγόριθμοι που θα περιγραφούν στην πτυχιακή εργασία. Μία σημαντική κλάση δέντρων είναι τα ριζωμένα δέντρα(*rooted trees*), στα οποία ένας κόμβος, που ονομάζεται ρίζα(*root*), είναι διακεκριμένος ενώ όλοι οι υπόλοιποι βρίσκονται σε διαδοχικά επίπεδα κάτω από αυτόν. Ένα ριζωμένο δέντρο που αποτελείται από 13 κόμβους φαίνεται στο παρακάτω σχήμα:



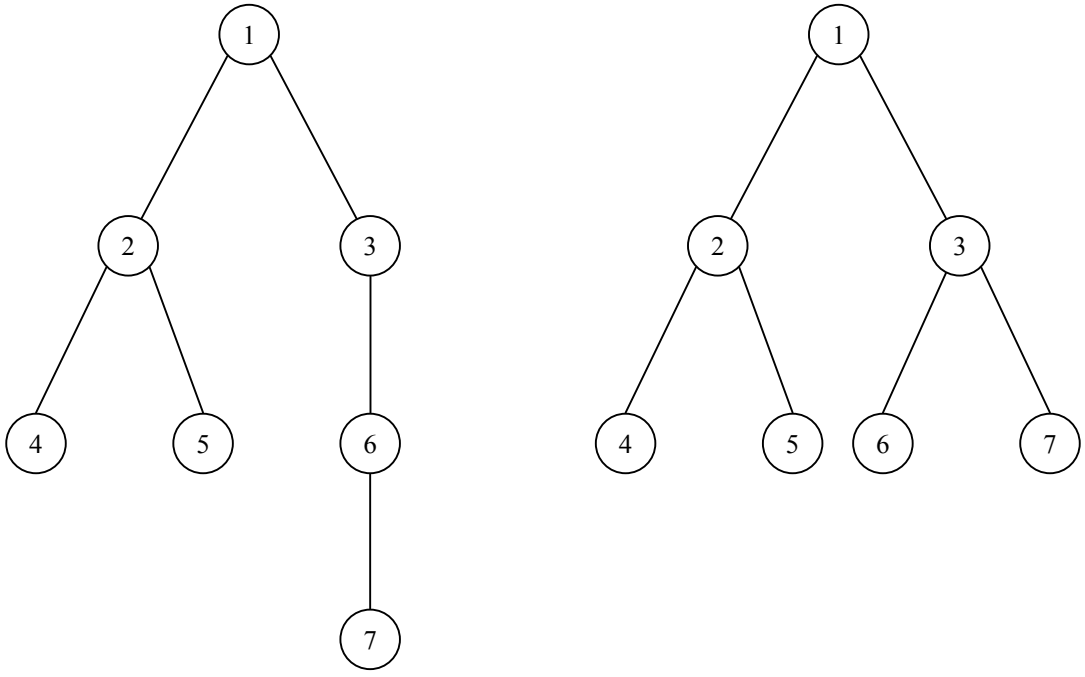
Εικόνα 1.2 - Ένα ριζωμένο δέντρο

Στη συνέχεια, όταν θα αναφερόμαστε στη λέξη δέντρο θα υπονοούμε ριζωμένο δέντρο. Επίσης, πρέπει να επισημανθεί ότι ένα σύνολο από ανεξάρτητα δέντρα ονομάζεται δάσος(*forest*).Για τα δέντρα χρησιμοποιείται η εξής ορολογία **[Lk]**:

- Αν x, y είναι δύο διακεκριμένες κορυφές ενός δέντρου τέτοιες ώστε η x να βρίσκεται στο μοναδικό μονοπάτι μεταξύ της ρίζας και της κορυφής y, τότε η x ονομάζεται γνήσιος πρόγονος της y και η y ονομάζεται γνήσιος απόγονος της κορυφής x.Αν επιπλέον υπάρχει η ακμή (x, y) τότε λέμε ότι πατέρας(*father*) της κορυφής y είναι η κορυφή x και ότι ένα από τα παιδιά(*childs*) της κορυφής x είναι η κορυφή y.Παραδείγματος χάριν οι κόμβοι 12,13 του σχήματος 1.2 είναι γνήσιοι

απόγονοι του κόμβου 3 ενώ ο κόμβος 12 είναι το μοναδικό παιδί του κόμβου 7. Σημειώστε ότι ένας κόμβος μπορεί να έχει μοναδικό πατέρα αλλά περισσότερα από ένα παιδιά.

- Ο μοναδικός κόμβος που δεν έχει πατέρα αποτελεί τη ρίζα του δέντρου.
- Όλοι οι κόμβοι που δεν έχουν παιδιά ονομάζονται τερματικοί κόμβοι ή φύλλα(*leaves*) του δέντρου.
- Οι κόμβοι που βρίσκονται μεταξύ της ρίζας και των φύλλων του δέντρου ονομάζονται εσωτερικοί κόμβοι(*internal nodes*).
- Ένα δέντρο ονομάζεται $k-$αδικό δέντρο, $k=2,3..$ όταν ο μέγιστος αριθμός των παιδιών των κόμβων του είναι k.Συνεπώς, το δέντρο στο σχήμα 1.3 είναι ένα τετραδικό δέντρο.
- Ένα δέντρο ονομάζεται πλήρες $k-$αδικό δέντρο, $k=2,3..$ όταν κάθε εσωτερικός κόμβος του, συμπεριλαμβανομένου της ρίζας έχει ακριβώς k παιδιά. Ένα από τα πιο συνηθισμένα δέντρα στο χώρο της Επιστήμης των Υπολογιστών είναι το δυαδικό δέντρο, το οποίο φαίνεται στο σχήμα 1.3.
- Βάθος ενός δέντρου h ονομάζεται το σύνολο των επιπέδων ενός δέντρου ξεκινώντας την αρίθμηση από το 0.
- Αποδεικνύεται ότι ένα πλήρες $k-$αδικό δέντρο, $k=2,3..$ βάθους h έχει συνολικά $\sum_{i=0}^{h} k^h$ κόμβους. Επίσης, αποδεικνύεται ότι για το ύψος h ενός $k-$αδικού δέντρου, $k=2,3..$ που αποτελείται από n κόμβους θα ισχύει $h \le \lceil \log_k(n) \rceil$.Η ισότητα ισχύει όταν το δέντρο είναι πλήρες.



Εικόνα 1.3 – Ένα δυαδικό δέντρο και ένα πλήρες δυαδικό δέντρο

1.2.4.2 Το Διάνυσμα Πατέρα Κόμβου

Μια πολύ σημαντική δομή δεδομένων που θα χρησιμοποιήσουμε για την αποθήκευση των δέντρων στους αλγορίθμους που θα περιγράψουμε είναι το διάνυσμα του πατέρα κόμβου p. Το διάνυσμα του πατέρα κόμβου αποτελεί έναν πολύ οικονομικό και εύχρηστο τρόπο αποθήκευσης και προκύπτει αν στη θέση i του διανύσματος αποθηκεύσουμε τον

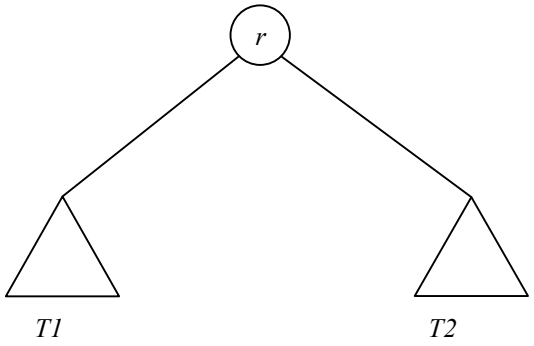
πατέρα του κόμβου i.Εξ ‘ ορισμού, αν r είναι η ρίζα του δέντρου, θέτουμε $p(r) = -1$. Συνεπώς το διάνυσμα πατέρα κόμβου(*father node vector*) που χρησιμοποιείται για την αποθήκευση του τετραδικού δέντρου του σχήματος 1.2 θα είναι το $p = [-1 \quad 1 \quad 1 \quad 1 \quad 2 \quad 2 \quad 3 \quad 3 \quad 3 \quad 3 \quad 6 \quad 7 \quad 10]$.

1.2.4.2 Το Διάνυσμα Βάθους

Συχνά είναι πολύ χρήσιμο να χρησιμοποιούμε και ένα διάνυσμα d το οποίο αποθηκεύει το βάθος κάθε κόμβου. Το διάνυσμα αυτό είναι απαραίτητο όπως θα δούμε για την ανανέωση ορισμένων δομών που χρησιμοποιούν οι αλγόριθμοι που θα περιγράψουμε στη συνέχεια. Μπορεί εύκολα να παραχθεί αναδρομικά, αν θέσουμε $d(r) = 0$ και $d(i) = d(p(i)) + 1$, $i \neq r$, όπου r η ρίζα του δέντρου και p το διάνυσμα πατέρα κόμβου. Συνεπώς το διάνυσμα d που αντιστοιχεί στο δέντρο του σχήματος 1.3 θα είναι το $d = [0 \quad 1 \quad 1 \quad 1 \quad 2 \quad 2 \quad 2 \quad 2 \quad 2 \quad 2 \quad 3 \quad 3 \quad 3]$. Εύκολα μπορεί να διαπιστώσει κανείς ότι το βάθος h του δέντρου συνδέεται με το διάνυσμα d με τη σχέση $h = \max\{d(i)\} \quad , i = 1,...,n$,όπου n το πλήθος των κόμβων του δέντρου.

1.2.4.3 Διάσχιση Δέντρων

Μια από τις πιο σημαντικές πράξεις επί των δέντρων με πάρα πολλές εφαρμογές είναι η διάσχιση τους. Διάσχιση(*traversal*) δέντρων είναι η διαδικασία εκείνη κατά την οποία οι κόμβοι εξετάζονται με μια σειρά συστηματικά έτσι ώστε κάθε κόμβος να επισκέπτεται μόνο μια φορά. Υπάρχουν 3 βασικά είδη διάσχισης ενός δέντρου. Θα παριγράψουμε την εφαρμογή των αναδρομικών αυτών διαδικασιών σε ένα δυαδικό δέντρο γενικής μορφής όπως αυτό που φαίνεται στο σχήμα 1.4 και μετά θα γενικεύσουμε τη προδιατεταγμένη διάσχιση(*preorder traversal*) για ένα $k-$αδικό δέντρο, $k = 2,3...$Στο σχήμα 1.4 με $T1$ και $T2$ συμβολίζουμε τα δυαδικά υπόδεντρα που συνδέονται με τη ρίζα του δέντρου.



Εικόνα 1.4 – Ένα δυαδικό δέντρο γενικής μορφής

Τα 3 βασικά είδη διάσχισης όπως ορίζονται για ένα δυαδικό δέντρο μπορούν να περιγραφούν από τις παρακάτω αναδρομικές διαδικασίες:

(i)Προδιατεταγμένη Διάσχιση(*Preorder Traversal*).

- Επίσκεψη της ρίζας r.
- Επίσκεψη του αριστερού υποδέντρου $T1$ εφαρμόζοντας προδιατεταγμένη διάσχιση.
- Επίσκεψη του δεξιού υποδέντρου $T2$ εφαρμόζοντας προδιατεταγμένη διάσχιση.

(ii)Ενδοδιατεταγμένη Διάσχιση(*Inorder Traversal*).

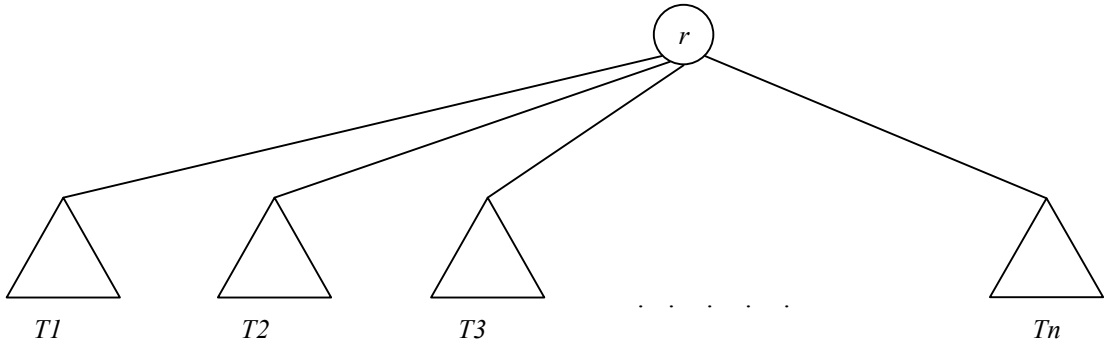
- Επίσκεψη του αριστερού υπόδεντρου $T1$ εφαρμόζοντας ενδοδιατεταγμένη διάσχιση.
- Επίσκεψη της ρίζας r.
- Επίσκεψη του δεξιού υποδέντρου $T2$ εφαρμόζοντας ενδοδιατεταγμένη διάσχιση.

(iii)Μεταδιατεταγμένη Διάσχιση(*Postorder Traversal*)

- Επίσκεψη του αριστερού υπόδεντρου $T1$ εφαρμόζοντας ενδοδιατεταγμένη διάσχιση.

- Επίσκεψη του δεξιού υποδέντρου $T2$ εφαρμόζοντας ενδοδιατεταγμένη διάσχιση.
- Επίσκεψη της ρίζας r.

Στους αλγορίθμους που θα περιγράψουμε θα χρησιμοποιήσουμε την προδιατεταγμένη διάσχιση δέντρων. Η γενίκευση της προδιατεταγμένης διάσχισης για ένα $k-$αδικό δέντρο, $k = 2,3..$ όπως αυτό του σχήματος 1.5 μπορεί να γίνει χρησιμοποιώντας την παρακάτω



Εικόνα 1.5 – Ένα *k*-αδικό δέντρο γενικής μορφής, *n*<=*k*

αναδρομική διαδικασία *Preorder(T),* όπου T το αρχικό $k-$αδικό δέντρο:

- Επίσκεψη της ρίζας r.
- Για κάθε παιδί Ti , $1 \le i \le n$ της ρίζας r εφαρμόζεται προδιατεταγμένη διάταξη, καλώντας την *Preorder(Ti)* .

Η κλήση της διαδικασίας *Preorder(T)* θα παράγει το λεγόμενο διάνυσμα της προδιατεταγμένης διάταξης που είναι ένα διάνυσμα που περιλαμβάνει όλους τους κόμβους του δέντρου με τη σειρά με την οποία αυτοί επισκέφτηκαν, εφαρμόζοντας προδιατεταγμένη διάσχιση. Συνεπώς το διάνυσμα προδιατεταγμένης διάσχισης που θα αντιστοιχεί στο τετραδικό δέντρο του σχήματος 1.2 θα είναι το:

$$y = [1 \quad 2 \quad 5 \quad 6 \quad 11 \quad 4 \quad 3 \quad 7 \quad 12 \quad 8 \quad 9 \quad 10 \quad 13]$$

Θεωρητικές έννοιες δέντρων καλύπτονται πλήρως στο βιβλίο του D.E. Knuth **[Kn]**.

1.3 ΟΡΙΣΜΟΙ ΠΡΟΒΛΗΜΑΤΩΝ ΚΑΙ ΜΑΘΗΜΑΤΙΚΟ ΥΠΟΒΑΘΡΟ

1.3.1 Το Πρόβλημα Μεταφοράς(*The Transportation Problem*)

Ένα από τα κλασικότερα προβλήματα που λύνει ο Δικτυακός Προγραμματισμός είναι το Πρόβλημα Μεταφοράς. Το Πρόβλημα Μεταφοράς αναφέρεται σε διμερή δίκτυα. Διμερές δίκτυο (*bipartite network*) $G = (N, A)$ είναι ένα κατευθυνόμενο γράφημα του οποίου οι κόμβοι έχουν χωριστεί σε δύο σύνολα N_1 και N_2 τέτοια ώστε $N = N_1 \cup N_2$, $N_1 \cap N_2 = \varnothing$ και $\forall (i, j) \in A \;\; i \in N_1$ και $j \in N_2$.

Έστω λοιπόν D και S τα δύο σύνολα κόμβων ενός διμερούς γραφήματος $G = (N, A)$. Όπως αναφέρθηκε παραπάνω θα είναι $N = S \cup D$ και $S \cap D = \varnothing$. Στο Πρόβλημα Μεταφοράς οι κόμβοι του συνόλου S ονομάζονται κόμβοι προσφοράς(*supply nodes*) ενώ οι κόμβοι του συνόλου D ονομάζονται κόμβοι ζήτησης (*demand nodes*) και όλα τα τόξα του γραφήματος κατευθύνονται από το S στο D. Σε κάθε κόμβο i του συνόλου S αντιστοιχούμε έναν αριθμό $a(i) > 0$, που αντιπροσωπεύει την ποσότητα ενός μεγέθους που

μπορεί να προσφερθεί από τον κόμβο i. Αντιστοίχως, σε κάθε κόμβο j του συνόλου D αντιστοιχούμε έναν αριθμό $b(j) > 0$ που αντιστοιχεί στην ποσότητα του ίδιου μεγέθους που απορροφά ο κόμβος j. Επίσης, με c_{ij} συμβολίζουμε το κόστος μεταφοράς μιας μονάδας του μεγέθους από τον κόμβο i του συνόλου S στον κόμβο j του συνόλου D. Παράλληλα, με x_{ij} συμβολίζουμε τις μονάδες του μεγέθους που μεταφέρονται από τον κόμβο i του συνόλου S στον κόμβο j του συνόλου D. Θεωρούμε ότι το σύνολο S έχει m κόμβους προσφοράς και ότι το σύνολο D έχει n κόμβους ζήτησης. Έτσι, μιλάμε για ένα $m \times n$ πρόβλημα μεταφοράς. Τέλος, υποθέτουμε ότι υπάρχει ένα τόξο (i, j) για κάθε ζευγάρι κόμβων i, j, δηλαδή έχουμε $m \times n$ τόξα. Οι αλγόριθμοι που θα παρουσιασθούν λύνουν ισοζυγισμένα προβλήματα, δηλαδή προβλήματα στα οποία η προσφορά είναι πάντα ίση με τη ζήτηση. Συνεπώς, θα ισχύει:

$$\sum_{i \in S} a(i) = \sum_{j \in D} b(j)$$

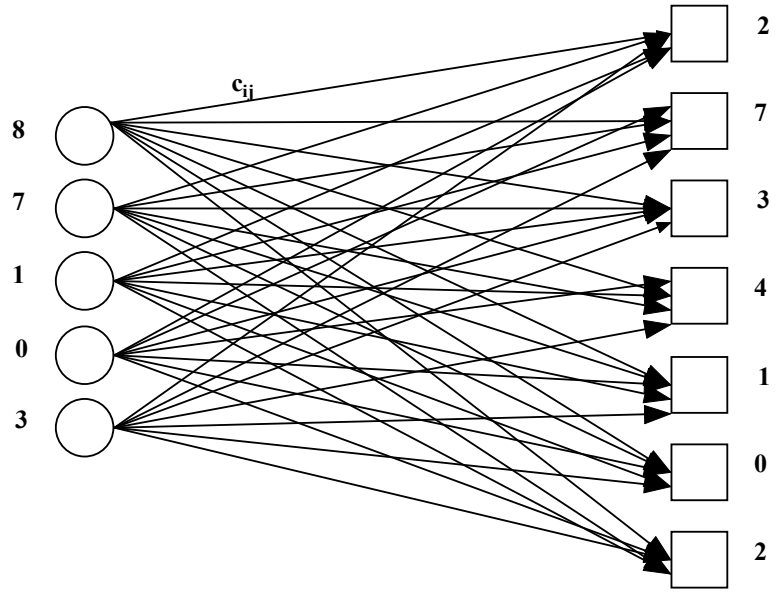
Το Πρόβλημα Μεταφοράς μπορεί τώρα να διατυπωθεί ως εξής:
«Ζητείται να βρεθεί ένα σχέδιο μεταφοράς $\sum_{i \in S} a(i)$ ***μονάδων από τους κόμβους προσφοράς στους κόμβους ζήτησης έτσι ώστε αυτή η μεταφορά να επιτευχθεί με το ελάχιστο κόστος.»***.
Η μαθηματική διατύπωση του Προβλήματος Μεταφοράς είναι η παρακάτω:

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

$$\mu.\pi. \quad \sum_{j=1}^{n} x_{ij} = a(i) \quad , i \in S \quad , \quad \sum_{i=1}^{m} x_{ij} = b(j) \quad , j \in D$$

$$x_{ij} \geq 0 \quad , i \in S \ , j \in D$$

Τα δεδομένα εισόδου του προβλήματος είναι ένας $m \times n$ πίνακας κόστους , ένα m - διάστατο διάνυσμα προσφοράς και ένα n - διάστατο διάνυσμα ζήτησης. Η λύση του προβλήματος είναι ένας $m \times n$ πίνακας που περιέχει τις μεταβλητές απόφασης x_{ij} .Η γραφική αναπαράσταση ενός 5×7 Προβλήματος Μεταφοράς φαίνεται στο σχήμα 1.6:



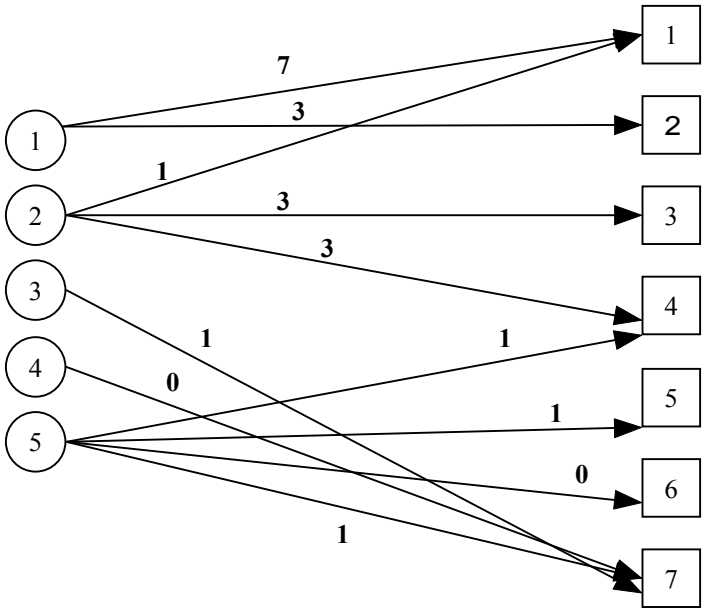
Εικόνα 1.6 – Γραφική αναπαράσταση ενός 5x7 Προβλήματος Μεταφοράς

Στο σχήμα 1.6 έχουμε αναπαραστήσει τους κόμβους ζήτησης με τετράγωνα και τους κόμβους προσφοράς με κύκλους έτσι ώστε να μπορούμε να τους διακρίνουμε. Οι τιμές δίπλα στους κόμβους του διμερούς γραφήματος αναφέρονται στις τιμές των στοιχείων των διανυσμάτων προσφοράς και ζήτησης. Για το σχήμα 1.6 συνεπώς είναι $a = [8 \quad 7 \quad 1 \quad 0 \quad 3]$, $b = [2 \quad 7 \quad 3 \quad 4 \quad 1 \quad 0 \quad 2]$ και τα μοναδιαία κόστη c_{ij} δίνονται από τη σχέση $c_{ij} = \frac{a(i) + b(j)}{2(i+j)}$, $i = 1..m$, $j = 1..n$.

Λύνοντας το συγκεκριμένο πρόβλημα με έναν από τους αλγορίθμους που θα περιγράψουμε παίρνουμε βέλτιστη τιμή αντικειμενικής συνάρτησης $z = 451.5$.Η λύση που παράγει ένας από τους αλγορίθμους που θα εξετάσουμε είναι ο πίνακας:

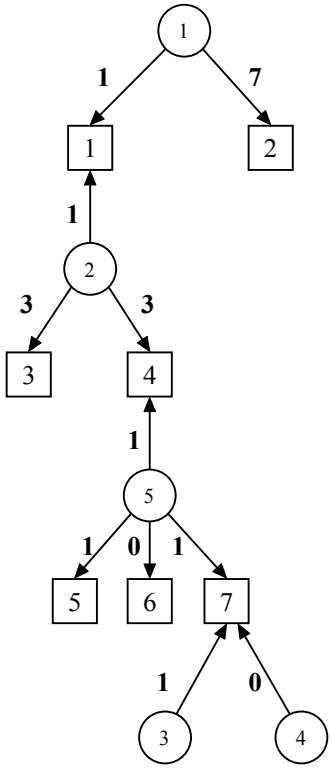
$$x = \begin{bmatrix} 1 & 7 & \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & 3 & 3 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 1 \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & 1 & 1 & 0 & 1 \end{bmatrix}$$

Ο παραπάνω πίνακας αντιστοιχεί στο παρακάτω διμερές γράφημα:



Εικόνα 1.7 – Βέλτιστη λύση του προβλήματος του σχήματος 1.7

Αυτό που πρέπει να παρατηρήσουμε στο παραπάνω σχήμα ότι το διμερές γράφημα της λύσης είναι δέντρο, αφού αποτελείται από *12 – 1 = 11* τόξα και είναι συνεκτικό. Αυτή η παρατήρηση είναι πολύ σημαντική αφού όπως θα δούμε και στη συνέχεια, οι αλγόριθμοι που θα παρουσιάσουμε επεξεργάζονται δέντρα. Δηλαδή, ξεκινούν από ένα εφικτό δέντρο που ικανοποιεί τους περιορισμούς του προβλήματος και σε κάθε επανάληψη κατασκευάζουν δέντρα που αντιστοιχούν σε καλύτερη τιμή αντικειμενικής συνάρτησης. Η οπτική παρουσίαση γίνεται με ριζωμένα δέντρα έτσι ώστε να μπορούμε να εκμεταλλευτούμε τις ιδιότητες που περιγράψαμε και για να διευκολυνθεί ο προγραμματισμός των αντίστοιχων αλγορίθμων που θα εξεταστούν. Το αντίστοιχο ριζωμένο δέντρο του σχήματος 1.7 είναι το παρακάτω(η ρίζα έχει επιλεγεί αυθαίρετα και η επιλογή αυτή δεν επηρεάζει την ορθότητα των αλγορίθμων.):



Εικόνα 1.8 – Το αντίστοιχο ριζωμένο βέλτιστο δέντρο για το διμερές γράφημα του σχήματος 1.8

Εδώ πρέπει να επισημάνουμε ότι το δέντρο της βέλτιστης λύσης που παράγεται από διαφορετικούς αλγορίθμους που λύνουν το ίδιο πρόβλημα δεν είναι πάντα το ίδιο. Αυτό σημαίνει ότι ένα διαφορετικός αλγόριθμος από αυτόν που χρησιμοποιήσαμε για να παράγουμε το βέλτιστο δέντρο του σχήματος 1.8 θα μπορούσε να παράγει ένα βέλτιστο δέντρο με τελείως διαφορετική δομή. Αυτό όμως που δεν αλλάζει είναι η βέλτιστη τιμή της αντικειμενικής συνάρτησης $z_{\min} = \sum_{(i,j)\in T} c_{ij}x_{ij}$, όπου T το βέλτιστο δέντρο, η οποία πρέπει να είναι πάντα η ίδια, ανεξάρτητα από τον αλγόριθμο που χρησιμοποιούμε.

Τελειώνοντας με τον ορισμό του Προβλήματος Μεταφοράς, θα αναφερθούμε στις συνθήκες βελτιστότητας του προβλήματος, δηλαδή τις συνθήκες εκείνες που πρέπει να ισχύουν έτσι ώστε το τρέχον δέντρο να είναι βέλτιστο. Αντιστοιχίζουμε σε κάθε κόμβο γραμμή $i \in S$ τη δυϊκή μεταβλητή u_i και σε κάθε κόμβο στήλη $j \in D$ τη δυϊκή μεταβλητή v_j . Θέτουμε τώρα:

$$s_{ij} = c_{ij} - u_i - v_j \ , i = 1,...,m \ , j = 1,...,n$$

Το τρέχον δέντρο T είναι βέλτιστο εάν και μόνον αν είναι εφικτό και $\forall (i,j) \notin T$ είναι $s_{ij} \geq 0$.

1.3.2 Το Πρόβλημα Αντιστοίχησης (*The Assignment Problem*)

Το Πρόβλημα Αντιστοίχησης ή όπως λέγεται πρόβλημα Ανάθεσης Έργων ή πρόβλημα Εκχώρησης αποτελεί ένα πολύ διαδεδομένο πρόβλημα στο χώρο της Επιστήμης των Υπολογιστών και της Επιχειρησιακής Έρευνας. Πολλοί αλγόριθμοι αναπτύσσονται συνεχώς για την επίλυσή του. Αποτελεί μια ειδική περίπτωση του προβλήματος Μεταφοράς από το οποίο προκύπτει θέτοντας $m = n$ και $a(i) = b(i) = 1$, $i = 1,...,n$. Ο ακέραιος n αποτελεί το μέγεθος ή τη διάσταση του προβλήματος που λύνουμε κάθε φορά.

Το Πρόβλημα Αντιστοίχησης βρίσκει πολλές εφαρμογές στην καθημερινή ζωή. Το κλασσικότερο παράδειγμα εφαρμογής του προβλήματος Αντιστοίχησης είναι η βέλτιστη ανάθεση n έργων σε n εργοδότες εάν μας είναι γνωστά τα κόστη εκτέλεσης:

του $1^{ου}$, $2^{ου}$,....., n -οστού έργου από τον $1^{ο}$ εργοδότη
του $1^{ου}$, $2^{ου}$,....., n -οστού έργου από τον $2^{ο}$ εργοδότη
.....
.....
.....
.....
.....
του $1^{ου}$, $2^{ου}$,....., n -οστού έργου από τον n -οστό εργοδότη

Ένα παράδειγμα από το χώρο της Επιστήμης των Υπολογιστών αποτελεί η βέλτιστη αξιοποίηση μιας ομάδας n επεξεργαστών (CPU) που απαιτούνται για την ταυτόχρονη λειτουργία n περιφερειακών συσκευών (εκτυπωτών, δίσκων) εάν ξέρουμε ότι η ανάθεση της συσκευής j στον επεξεργαστή i απαιτεί περιφερειακή μνήμη R_{ij} MB, $1 \le i, j \le n$.

Παράλληλα, η επίλυση του προβλήματος Αντιστοίχησης βρίσκει εφαρμογή και στα Δίκτυα Ηλεκτρονικών Υπολογιστών όπου προσπαθούμε να βρούμε ένα βέλτιστο τρόπο για να συνδέσουμε συγκεντρωντές(*concentrators*) με κόμβους(*nodes*) ενός δικτύου.
Στο Πρόβλημα Αντιστοίχησης χρησιμοποιούμε την ίδια γραφική αναπαράσταση με το πρόβλημα Μεταφοράς. Συμβολίζουμε με x_{ij} τη μεταβλητή απόφασης που αντιστοιχεί στο τόξο (i, j) η οποία είναι ίση με τη μονάδα, αν το έργο i ανατίθεται στον εργοδότη j, αλλιώς είναι ίση με το μηδέν .Το Πρόβλημα Αντιστοίχησης μπορεί τώρα να διατυπωθεί ως εξής:
«Ζητείται να βρεθεί η βέλτιστη ανάθεση n ***εργασιών – αντικειμένων σε*** n ***οντότητες – αντικείμενα έτσι ώστε το συνολικό κόστος να είναι ελάχιστο .».***
Η μαθηματική διατύπωση του Προβλήματος Αντιστοίχησης είναι η παρακάτω:

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

$$\mu.\pi. \quad \sum_{i=1}^{n} x_{ij} = 1 \ , \ j = 1,...,n \ , \ \sum_{j=1}^{n} x_{ij} = 1 \ , \ i = 1,...,n \ ,$$

$$0 \le x_{ij} \le 1 \ , 1 \le \ i \ , j \ \le n \ .$$

Τα δεδομένα εισόδου του προβλήματος είναι μόνο ένας $n \times n$ πίνακας κόστους. Οι συνθήκες βελτιστότητας του προβλήματος είναι ίδιες με αυτές του προβλήματος Μεταφοράς.

Εύκολα βέβαια θα μπορούσε κανείς να συμπεράνει ότι το πρόβλημα Αντιστοίχησης μπορεί να λυθεί πολύ εύκολα αν συγκρίνουμε όλα τα πιθανά κόστη που προκύπτουν και επιλέξουμε το μικρότερο. Αν σκεφτεί όμως κάποιος ότι υπάρχουν $n!$ μεταθέσεις n διακεκριμένων αντικειμένων, συμπεραίνουμε ότι το κόστος αυτής της διαδικασίας είναι απαγορευτικό και ως εκ τούτου οδηγούμαστε σε συνδυαστική έκρηξη . Εδώ βασικά είναι το σημείο που φαίνεται η αξία των αλγορίθμων της συνδυαστικής βελτιστοποίησης, οι οποίοι λύνουν το πρόβλημα σε πολυωνυμικό χρόνο $O(n^3)$ και όχι σε χρόνο τάξης $O(n!)$.

Εκτενή αναφορά σε προβλήματα και αλγορίθμους συνδυαστικής βελτιστοποίησης γίνεται στο βιβλίο**[PSt]**.

1.4 ΑΝΑΦΟΡΕΣ

[Kn] D.E. Knuth., *"The Art of Computer Programming – Fundamental Algorithms"*, 3rd edition, Addison-Wesley, 1997

[PSt] C. Papadimitriou, K. Steiglitz "*Combinatorial Optimization: Algorithms and Complexity"*, Prentice Hall (1982)

[Sdg] R. Sedgewick, "*Algorithms in C, Parts 1-4"* Third Edition, Addison Wesley, 1998

[CLL] C.L. Liu, "*Elements of Discrete Mathematics*", Mc-Graw Hill Inc., 1985

[SS] Stephanides G., Samaras N., *"Computational Methods with Matlab"*, Zygos Publications, Thessaloniki 1999, in Greek

[PK] Paparrizos K., *"Matlab 6"*, Zygos Publications, Thessaloniki 2001, in Greek

[Lk] M. Loukakis, "*Data Structures-Algorithms"* Zygos Publications, Thessaloniki 1998, in Greek

[PP] C. Papamanthou, K. Paparrizos, *"A Visualization of the Primal Simplex Algorithm for the Assignment Problem"*, accepted demo proposal, ITiCSE 2003

[PPS] C. Papamanthou, K. Paparrizos, N. Samaras, *"An Educational Visualization Software for the Assignment Problem"*, submitted for publication (PCI 2003 – Panhellenic Conference in Informatics)

ΚΕΦΑΛΑΙΟ 2

Ο ΠΡΩΤΕΥΩΝ ΑΛΓΟΡΙΘΜΟΣ SIMPLEX

Ο πρωτεύον αλγόριθμος Simplex είναι ένας κλασσικός αλγόριθμος Γραμμικού Προγραμματισμού που εφευρέθηκε το 1947 από τον G.B.Danzig**[Dn1]** για την επίλυση του γενικού γραμμικού προβλήματος. Συγκαταλέγεται στις δέκα μεγαλύτερες ανακαλύψεις στο χώρο της Επιστήμης των Υπολογιστών[1] μαζί με αλγορίθμους όπως ο QuickSort και ο FFT(*Fast Fourier Transform*). Ο πρωτεύων αλγόριθμος Simplex για τα προβλήματα Μεταφοράς και Αντιστοίχισης[2] όπως θα παρουσιασθεί εδώ αποτελεί στην ουσία μια εξειδίκευση του αλγορίθμου Simplex για το γενικό γραμμικό πρόβλημα. Δηλαδή, μπορεί να αποδειχθεί, ότι ο αλγόριθμος που θα παρουσιασθεί είναι ακριβώς ο ίδιος αλγόριθμος που χρησιμοποιείται για το γραμμικό πρόβλημα, προσαρμοσμένος στα δίκτυα.

2.1 ΕΦΑΡΜΟΓΗ ΣΤΟ ΠΡΟΒΛΗΜΑ ΜΕΤΑΦΟΡΑΣ

Στη συνέχεια θα περιγράψουμε πως μπορεί ο πρωτεύων αλγόριθμος Simplex να εφαρμοστεί για τη επίλυση του προβλήματος Μεταφοράς. Θα παραθέσουμε την περιγραφή του αλγορίθμου, τη βηματική του εκτέλεση σε μορφή ψευδοκώδικα καθώς και λεπτομέρειες για έναν αποτελεσματικό τρόπο προγραμματισμού του αλγορίθμου.

2.1.1 Υπολογισμός ενός Εφικτού Δέντρου Ξεκινήματος

Για να ξεκινήσει η επίλυση ενός προβλήματος Μεταφοράς, χρειάζεται, όπως και σε κάθε πρόβλημα Γραμμικού Προγραμματισμού, να προσδιοριστεί μια αρχική εφικτή λύση. Στην περίπτωσή μας, πρέπει να υπολογίσουμε ένα αρχικό ισχυρό εφικτό δέντρο ξεκινήματος. Υπάρχουν διάφοροι τρόποι υπολογισμού του εφικτού δέντρου ξεκινήματος. Εμείς θα περιγράψουμε δύο μεθόδους, τη μέθοδο VAM (*Vogel's Approximation Method*) και τη μέθοδο της βορειοδυτικής γωνίας (*North West Corner Method*), εκ των οποίων θα χρησιμοποιήσουμε τη δεύτερη.

2.1.1.1 Η μέθοδος VAM

Η μέθοδος VAM χρησιμοποιεί το κόστος Μεταφοράς ως κριτήριο για τον προσδιορισμό του εφικτού δέντρου ξεκινήματος. Έστω λοιπόν έχουμε να λύσουμε ένα $m \times n$ πρόβλημα Μεταφοράς με δεδομένα έναν $m \times n$ πίνακα κόστους C, ένα m- διάστατο διάνυσμα προσφοράς A και ένα n-διάστατο διάνυσμα ζήτησης B.Ο αλγόριθμος υπολογισμού ενός εφικτού δέντρου ξεκινήματος με τη μέθοδο VAM δουλεύει ως εξής:

Βήμα 0

Υπολογίζονται οι διαφορές d_i και d_j μεταξύ του ελαχίστου και επόμενου ελαχίστου κόστους κάθε σειράς i και κάθε στήλης j του πίνακα κόστους, οι οποίες δεν έχουν διαγραφεί.

Βήμα 1

Θέτουμε $d_k = \max\left\{\max\{d_i,\ i=1,...,m\}, \max\left\{d_j,\ j=1,...,n\right\}\right\}$.

Βήμα 2

Αν k είναι γραμμή του πίνακα κόστους τότε υπολογίζουμε το ελάχιστο κόστος της συγκεκριμένης γραμμής του πίνακα και συνεπώς θέτουμε $C_{rt} = \min\{C_{kj}\},\ \ j=1,...,n$ αλλιώς, δηλαδή αν k είναι στήλη του πίνακα κόστους, θέτουμε αντίστοιχα $C_{rt} = \min\{C_{ik}\},\ \ i=1,...,m$.

---

[1] *"If one would take statistics about which mathematical problem is using up most of the computer time in the world, then the answer would probably be linear programming"*, Laszlo Lovasz, 1980

[2] Ο αλγόριθμος Simplex για προβλήματα αντιστοίχησης αναπτύχθηκε ξεχωριστά και από τους Barr, Glover και Klingman, δες **[BGK]**.

Βήμα 3

αλλιώς($A_r \le B_t$) θέτουμε $e = X_{rt} = A_r$, $B_t' = B_t - e$, και διαγράφουμε τη γραμμή r . Το τόξο (r,t) αποτελεί ένα βασικό τόξο του εφικτού δέντρου ξεκινήματος με e μονάδες μεταφοράς.

Βήμα 4:

Αν έχουν διαγραφεί όλα τα κελιά του πίνακα, τότε stop, αλλιώς επέστρεψε στο βήμα 0.

Παράδειγμα 2.1

Δίνεται ο πίνακας κόστους

$$C = \begin{bmatrix} -5 & 49 & 28 & 20 & 47 & 43 \\ 19 & 62 & 32 & 61 & 0 & 13 \\ -2 & 63 & 14 & 46 & 28 & 53 \\ 1 & 50 & 41 & 26 & 59 & 0 \end{bmatrix}$$

τα διανύσματα προσφοράς και ζήτησης $a = \begin{bmatrix} 5 & 2 & 7 & 6 \end{bmatrix}$ ***και*** $b = \begin{bmatrix} 2 & 3 & 7 & 2 & 1 & 5 \end{bmatrix}$ ***αντίστοιχα για ένα*** 4×6 ***πρόβλημα Μεταφοράς. Να προσδιορισθεί το εφικτό δέντρο ξεκινήματος χρησιμοποιώντας τη μέθοδο VAM.***

Λύση:

Κατασκευάζουμε τον πίνακα που περιέχει τα κόστη, τις διαφορές κάθε γραμμής και κάθε στήλης καθώς και τις τιμές των διανυσμάτων a, b :

Πίνακας 2.1

d_i, d_j	3	1	14	6	28	13	a
25	*-5*	*49*	*28*	*20*	*47*	*43*	**5**
13	*19*	*62*	*32*	*61*	*0*	*13*	**2**
16	*-2*	*63*	*14*	*46*	*28*	*53*	**7**
1	*1*	*50*	*41*	*26*	*59*	*0*	**6**
b	**2**	**3**	**7**	**2**	**1**	**5**	

Είναι $d_k = \max\{d_i, d_j\} = 28$. Συνεπώς ψάχνουμε να βρούμε το ελάχιστο κόστος στη στήλη 5, το οποίο βρίσκεται στο κελί $(2,5)$. Άρα, το πρώτο βασικό τόξο του δέντρου που προκύπτει θα είναι το $(2,5)$ με $x_{25} = 1$ μονάδα μεταφοράς ($A_2 = 1 < B_5$).Επίσης, ανανεώνεται η διαφορά μόνο της δεύτερης γραμμής, άρα θέτουμε $d_2 = 6$ και το δεύτερο στοιχείο του διανύσματος της προσφοράς το οποίο τίθεται ίσο με 2 – 1 = 1. Τα ανανεωμένα δεδομένα φαίνονται παρακάτω:

Πίνακας 2.2

d_i, d_j	3	1	14	6		13	a
25	*-5*	*49*	*28*	*20*		*43*	**5**
6	*19*	*62*	*32*	*61*	**1**	*13*	**1**
16	*-2*	*63*	*14*	*46*		*53*	**7**
1	*1*	*50*	*41*	*26*		*0*	**6**
b	**2**	**3**	**7**	**2**		**5**	

Είναι $d_k = \max\{d_i, d_j\} = 25$. Συνεπώς ψάχνουμε να βρούμε το ελάχιστο κόστος στη γραμμή 1 το οποίο βρίσκεται στο κελί $(1,1)$. Άρα, το δεύτερο βασικό τόξο του δέντρου που προκύπτει θα είναι το $(1,1)$ με $x_{11} = 2$ μονάδες μεταφοράς($B_1 = 2 < A_1$) και τα ανανεωμένα δεδομένα φαίνονται παρακάτω:

Πίνακας 1.3

d_i, d_j		1	14	6		13	a
8	**2**	*49*	*28*	*20*		*43*	**3**
19		*62*	*32*	*61*	**1**	*13*	**1**
32		*63*	*14*	*46*		*53*	**7**
26		*50*	*41*	*26*		*0*	**6**
b		**3**	**7**	**2**		**5**	

Είναι $d_k = \max\{d_i, d_j\} = 32$. Συνεπώς ψάχνουμε να βρούμε το ελάχιστο κόστος στη γραμμή 3, το οποίο βρίσκεται στο κελί $(3,3)$. Άρα, το τρίτο βασικό τόξο του δέντρου που προκύπτει θα είναι το $(3,3)$ με $x_{33} = 7$ μονάδες μεταφοράς($B_3 = A_3 = 7$,άρα διαγράφεται η γραμμή 3 δες πίνακα 2.4). Με την ίδια λογική προκύπτουν και οι πίνακες κάτω από τον 2.4.

Πίνακας 2.4

d_i, d_j		1	4	6		13	a
8	**2**	*49*	*28*	*20*		*43*	**3**
19		*62*	*32*	*61*	**1**	*13*	**1**
			7				
26		*50*	*41*	*26*		*0*	**6**
b		**3**	**0**	**2**		**5**	

d_i, d_j		1	4	6			a
8	**2**	*49*	*28*	*20*			**3**
29		*62*	*32*	*61*	**1**		**1**
			7				
15		*50*	*41*	*26*		**5**	**1**
b		**3**	**0**	**2**			

d_i, d_j		1		6			a
29	**2**	*49*		*20*			**3**
1		*62*	**0**	*61*	**1**		**1**
			7				
24		*50*		*26*		**5**	**1**
b		**3**		**2**			

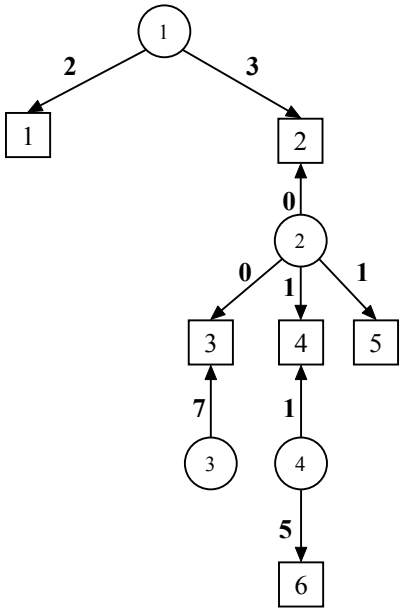
d_i, d_j		8		35			a
	2	**3**					
1		*62*	**0**	*61*	**1**		**1**
			7				
24		*50*		*26*		**5**	**1**
b		**0**		**2**			

d_i, d_j		62		61			a
	2	**3**					
1		*62*	**0**	*61*	**1**		**1**
			7				
				1		**5**	
b		**0**		**1**			

d_i, d_j				61			a
	2	**3**					
61		**0**	**0**	*61*	**1**		**1**
			7				
				1		**5**	
b				**1**			

d_i, d_j				61			a
	2	**3**					
		0	**0**	**1**	**1**		
			7				
				1		**5**	
b				**0**			

Το αντίστοιχο εφικτό δέντρο ξεκινήματος που αντιστοιχεί στον τελευταίο από τους παραπάνω πίνακες, θα είναι το παρακάτω:



Εικόνα 2.0 – Το εφικτό δέντρο ξεκινήματος κατασκευασμένο με τη μέθοδο VAM

2.1.1.2 Η μέθοδος της βορειοδυτικής γωνίας

Μια από τις πιο κλασσικές μεθόδους για τον υπολογισμό ενός αρχικού εφικτού δέντρου για το πρόβλημα Μεταφοράς όταν χρησιμοποιούμε τον αλγόριθμο Simplex είναι η μέθοδος της βορειοδυτικής γωνίας(*northwest corner method*)**[Dn2]**.

Η μέθοδος της βορειοδυτικής γωνίας κατασκευάζει ένα εφικτό ριζωμένο δέντρο με ρίζα συνήθως τον κόμβο γραμμή 1. Κατά τη κατασκευή αυτού του δέντρου χρησιμοποιούμε σαν δεδομένα μόνο τα διανύσματα προσφοράς και ζήτησης. Έστω λοιπόν a είναι το m-διάστατο διάνυσμα προσφοράς και b είναι το n-διάστατο διάνυσμα ζήτησης του προβλήματος που θέλουμε να λύσουμε. Ο σκοπός μας είναι ο υπολογισμός των τιμών των μεταβλητών απόφασης x_{ij} που αντιστοιχούν στα τόξα του αρχικού εφικτού δέντρου ξεκινήματος. Οι μεταβλητές απόφασης αποθηκεύονται σε έναν $m \times n$ πίνακα x όπου αρχικά οι τιμές όλων των στοιχείων του τίθενται ίσες με άπειρο, που υποδηλώνει την ανυπαρξία του συγκεκριμένου τόξου. Επίσης, δεξιά της γραμμής i του πίνακα x αναγράφονται για ευκολία οι τιμές $a(i)$, $i = 1,...,m$ ενώ κάτω από κάθε στήλη j του πίνακα αναγράφονται οι τιμές $b(j)$, $j = 1,...,n$. Η μέθοδος της βορειοδυτικής γωνίας δουλεύει ως εξής: Ξεκινούμε από το κελί $(1,1)$ του πίνακα x και θέτουμε $x_{11} = \min\{a(1), b(1)\}$. Μόλις προσδιορίσαμε τη τιμή της μεταβλητής απόφασης του πρώτου τόξου του εφικτού δέντρου ξεκινήματος. Στη συνέχεια διακρίνουμε τις εξής περιπτώσεις:

- Αν $a(1) < b(1)$, τότε έχουμε $x_{11} = a(1)$ και θέτουμε $a'(1) = a(1) - x_{11} = 0$ ενώ $b'(1) = b(1) - x_{11} = b(1) - a(1) > 0$. Στη συνέχεια μετακινούμαστε κάθετα στο κελί $(2,1)$ και θέτουμε πάλι $x_{21} = \min\{a(2), b'(1)\} = \min\{a(2), b(1) - a(1)\}$. Έτσι προστίθεται στο δέντρο το τόξο $(2,1)$.
- Αν $a(1) > b(1)$, τότε έχουμε $x_{11} = b(1)$ και θέτουμε $b'(1) = b(1) - x_{11} = 0$ ενώ $a(1) = a(1) - x_{11} = a(1) - b(1) > 0$. Στη συνέχεια μετακινούμαστε οριζόντια στο κελί $(1,2)$ και θέτουμε πάλι $x_{12} = \min\{a'(1), b(2)\} = \min\{a(1) - b(1), b(2)\}$. Έτσι προστίθεται στο δέντρο το τόξο $(1,2)$.

- Αν $a(1)=b(1)$, τότε θέτουμε $x_{22}=\min\{a(2),b(2)\}$ και επίσης θέτουμε $x_{12}=0$ έτσι ώστε να κατασκευαστεί ισχυρό δέντρο.

Συνοψίζοντας, θα περιγράψουμε μια γενική επανάληψη του αλγορίθμου για τον υπολογισμό του εφικτού δέντρου με τη μέθοδο της βορειοδυτικής γωνίας. Έστω λοιπόν είμαστε στο κελί (i,j). Όλες οι εξισώσεις που αντιστοιχούν σε κόμβους γραμμές $k<i$ έχουν ικανοποιηθεί και όλες οι εξισώσεις που αντιστοιχούν σε κόμβους στήλες $\ell<j$ έχουν επίσης ικανοποιηθεί. Στη συνέχεια θέτουμε $x_{ij}=\min\{a(i),b(j)\}$ και ανανεώνονται οι τιμές των $a(i)$ και $b(j)$, θέτοντας $a'(i)=a(i)-x_{ij}$ και $b'(j)=b(j)-x_{ij}$. Επιπρόσθετα:

- Αν $a'(i)=0$ και $b'(j)>0$,μετακινούμαστε κάθετα στο κελί $(i+1,j)$.
- Αν $a'(i)>0$ και $b'(j)=0$, μετακινούμαστε οριζόντια στο κελί $(i,j+1)$.
- Αν $a'(i)=0$ και $b'(j)=0$, μετακινούμαστε διαγώνια στο κελί $(i+1,j+1)$, αφού πρώτα προσθέσουμε το τόξο $(i,j+1)$, θέτοντας $x_{i,j+1}=0$ για να εξασφαλίσουμε τη συνεκτικότητα του δέντρου.

Παράδειγμα 2.2

Δίνονται το διάνυσμα προσφοράς $a=\begin{bmatrix}5 & 2 & 7 & 6\end{bmatrix}$***, το διάνυσμα ζήτησης*** $b=\begin{bmatrix}2 & 3 & 7 & 2 & 1 & 5\end{bmatrix}$***, και ο πίνακας κόστους***

$$C=\begin{bmatrix}-5 & 49 & 28 & 20 & 47 & 43\\ 19 & 62 & 32 & 61 & 0 & 13\\ -2 & 63 & 14 & 46 & 28 & 53\\ 1 & 50 & 41 & 26 & 59 & 0\end{bmatrix}$$

για ένα 4×6 ***πρόβλημα Μεταφοράς. Να κατασκευασθεί ένα ισχυρό δέντρο ξεκινήματος με τη μέθοδο της βορειοδυτικής γωνίας.***

Λύση:

Ο πίνακας 2.5 είναι ο αρχικός. (Τα στοιχεία των διανυσμάτων a, b φαίνονται με έντονα γράμματα στις άκρες του πίνακα. Το σκιασμένο κελί είναι το τρέχον).

Πίνακας 2.5

∞	∞	∞	∞	∞	∞	**5**
∞	∞	∞	∞	∞	∞	**2**
∞	∞	∞	∞	∞	∞	**7**
∞	∞	∞	∞	∞	∞	**6**
2	**3**	**7**	**2**	**1**	**5**	

Θέτουμε $x_{11}=\min\{a(1),b(1)\}=\min\{5,2\}=2=b(1)$ και στη συνέχεια σύμφωνα με όσα αναφέρθηκαν θα έχουμε $b'(1)=0$, $a'(1)=a(1)-b(1)=5-2=3$ και μεταφερόμαστε στο κελί $(1,2)$.Συνεπώς, ο πίνακας 2.5 θα πάρει την παρακάτω μορφή:

Πίνακας 2.6

2	∞	∞	∞	∞	∞	**3**
∞	∞	∞	∞	∞	∞	**2**
∞	∞	∞	∞	∞	∞	**7**
∞	∞	∞	∞	∞	∞	**6**
0	**3**	**7**	**2**	**1**	**5**	

Στη συνέχεια επειδή $a(1)=b(2)=3$, θέτουμε $x_{12}=3$, $x_{13}=0$, $a'(1)=b'(2)=3$ και μεταφερόμαστε στο κελί $(2,3)$.Συνεπώς ο πίνακας 2.6 θα πάρει την παρακάτω μορφή:

Πίνακας 2.7

2	3	0	∞	∞	∞	**0**
∞	∞	∞	∞	∞	∞	**2**
∞	∞	∞	∞	∞	∞	**7**
∞	∞	∞	∞	∞	∞	**6**
0	**0**	**7**	**2**	**1**	**5**	

Συνεχίζοντας έχουμε $x_{23}=\min\{a(2),b(3)\}=\min\{2,7\}=2=a(2)$, άρα θα είναι $a'(2)=0$, $b'(3)=b(3)-a(2)=7-2=5$ και μεταφερόμαστε στο κελί $(3,3)$.Συνεπώς, ο πίνακας 2.7 θα πάρει την παρακάτω μορφή:

Πίνακας 2.8

2	3	0	∞	∞	∞	**0**
∞	∞	2	∞	∞	∞	**0**
∞	∞	∞	∞	∞	∞	**7**
∞	∞	∞	∞	∞	∞	**6**
0	**0**	**5**	**2**	**1**	**5**	

Με τον ίδιο τρόπο θέτουμε $x_{33}=\min\{a(3),b(3)\}=\min\{7,5\}=5=b(3)$ και στη συνέχεια σύμφωνα θα έχουμε $b'(3)=0$, $a'(3)=a(3)-b(3)=7-5=2$ και μεταφερόμαστε στο κελί $(3,4)$.Συνεπώς, ο πίνακας 2.8 θα πάρει την παρακάτω μορφή:

Πίνακας 2.9

2	3	0	∞	∞	∞	**0**
∞	∞	2	∞	∞	∞	**0**
∞	∞	5	∞	∞	∞	**2**
∞	∞	∞	∞	∞	∞	**6**
0	**0**	**0**	**2**	**1**	**5**	

Η ανανέωση του παραπάνω πίνακα γίνεται ως εξής: Επειδή $a(3)=b(4)=2$, θέτουμε $x_{34}=2$, $x_{35}=0$, $a'(3)=b'(4)=2$ και μεταφερόμαστε στο κελί $(4,5)$.Συνεπώς ο πίνακας 2.9 θα πάρει την παρακάτω μορφή:

Πίνακας 2.10

2	3	0	∞	∞	∞	**0**
∞	∞	2	∞	∞	∞	**0**
∞	∞	5	2	0	∞	**0**
∞	∞	∞	∞	∞	∞	**6**
0	**0**	**0**	**0**	**1**	**5**	

Στην επόμενη επανάληψη και ακολουθώντας ακριβώς την ίδια διαδικασία όπως προηγούμενα, θα έχουμε $x_{45}=\min\{a(4),b(5)\}=\min\{1,6\}=1=a(4)$ και συνεπώς $b'(5)=0$, $a'(4)=a(4)-b(5)=6-1=5$. Άρα μεταφερόμαστε στο κελί $(4,6)$ και συνεπώς ο πίνακας 2.10 θα πάρει την παρακάτω μορφή:

Πίνακας 2.11

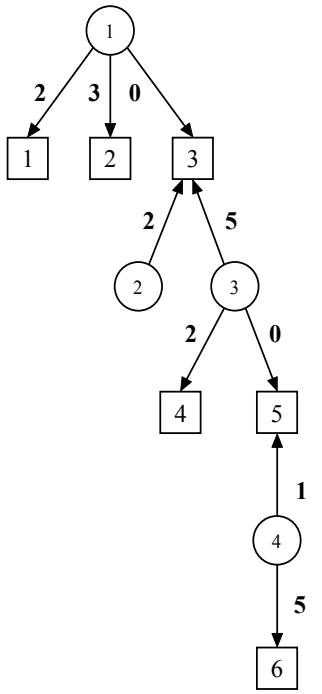
2	3	0	∞	∞	∞	**0**
∞	∞	2	∞	∞	∞	**0**
∞	∞	5	2	0	∞	**0**
∞	∞	∞	∞	1	∞	**5**
0	**0**	**0**	**0**	**0**	**5**	

Εκτελώντας και την τελευταία επανάληψη, όπου θέτουμε $x_{46} = 5$ και $a'(4) = b'(6) = 0$ ο πίνακας 2.11 θα πάρει την παρακάτω τελική μορφή :

Πίνακας 2.12

2	3	0	∞	∞	∞	**0**
∞	∞	2	∞	∞	∞	**0**
∞	∞	5	2	0	∞	**0**
∞	∞	∞	∞	1	5	**0**
0	**0**	**0**	**0**	**0**	**0**	

Σε αυτό το σημείο έχει πλέον προσδιορισθεί το ριζωμένο εφικτό δέντρο ξεκινήματος του προβλήματος. Δηλαδή ο πίνακας 2.12 αντιστοιχεί στο δέντρο το παρακάτω σχήματος:



Εικόνα 2.1 – Το εφικτό δέντρο ξεκινήματος του προβλήματος

Από τη στιγμή που έχει προσδιορισθεί το εφικτό δέντρο ξεκινήματος, μπορούμε πλέον να εφαρμόσουμε τον αλγόριθμο Simplex για την επίλυση του προβλήματος, ξεκινώντας με τη λύση που προσδιορίσαμε με τη βοήθεια της μεθόδου της βορειοδυτικής γωνίας. Μπορούμε, άνετα να επαληθεύσουμε ότι το δέντρο του σχήματος 2.1 είναι εφικτό αφού ικανοποιεί τους αρχικούς περιορισμούς του προβλήματος μεταφοράς.

Στη συνέχεια, και πριν αρχίσουμε την περιγραφή του αλγορίθμου Simplex, πρέπει να υπολογίσουμε και τα διανύσματα των δυϊκών μεταβλητών των κόμβων του εφικτού δέντρου.

Από τη δυϊκή θεωρία είναι γνωστό ότι το μειωμένο κόστος κάθε βασικής μεταβλητής ενός προβλήματος είναι ίση με το 0. Συνεπώς $\forall(i,j)\in T$, όπου T το εφικτό δέντρο του προβλήματος θα ισχύει:

$$s_{ij}(T) = c_{ij} - u_i(T) - v_j(T) = 0$$

Έχοντας $m+n-1$ βασικά τόξα και $m+n$ μεταβλητές ($u_i, i=1,...,m,\quad v_j, j=1,...,n$) μπορεί να προκύψει ένα γραμμικό σύστημα $m+n-1$ εξισώσεων με $m+n$ αγνώστους, αφού κάθε τόξο του δέντρου αντιστοιχεί και σε μια εξίσωση. Επίσης η δυϊκή μεταβλητή της ρίζας τίθεται εξ'ορισμού ίση με το μηδέν ($u_1 = 0$) και τελικά το γραμμικό σύστημα μετατρέπεται σε ένα $(m+n)\times(m+n)$ σύστημα, το οποίο λύνουμε πολύ εύκολα λόγω της ιδιαίτερης δομής του εφικτού δέντρου και βρίσκουμε τις τιμές των δυϊκών μεταβλητών. Το σύστημα των εξισώσεων που προκύπτουν για το εφικτό δέντρο που υπολογίσαμε θα είναι λοιπόν:

$$\left.\begin{aligned}
u_1 &= 0\\
c_{11} &= u_1 + v_1\\
c_{12} &= u_1 + v_2\\
c_{13} &= u_1 + v_3\\
c_{23} &= u_2 + v_3\\
c_{33} &= u_3 + v_3\\
c_{34} &= u_3 + v_4\\
c_{35} &= u_3 + v_5\\
c_{45} &= u_4 + v_5\\
c_{46} &= u_4 + v_6
\end{aligned}\right\}$$

Αντικαθιστώντας τα c_{ij}, βρίσκουμε εύκολα τη λύση που είναι τα εξής διανύσματα $u = \begin{bmatrix}0 & 4 & -14 & 17\end{bmatrix}$ και $v = \begin{bmatrix}-5 & 49 & 28 & 60 & 42 & -17\end{bmatrix}$. Στη συνέχεια υπολογίζουμε τα μειωμένα κόστη των μη βασικών τόξων του δέντρου, χρησιμοποιώντας τον τύπο $s_{ij} = c_{ij} - u_i - v_j$ και υπολογίζουμε έναν πίνακα s που τον ονομάζουμε πίνακα των μειωμένων κοστών και είναι από τις πιο σημαντικές δομές δεδομένων που θα επεξεργάζεται ο αλγόριθμος που θα περιγραφεί. Συνεπώς, κάνοντας πράξεις, βρίσκουμε ότι για το συγκεκριμένο πρόβλημα αυτός ο πίνακας θα είναι ο παρακάτω:

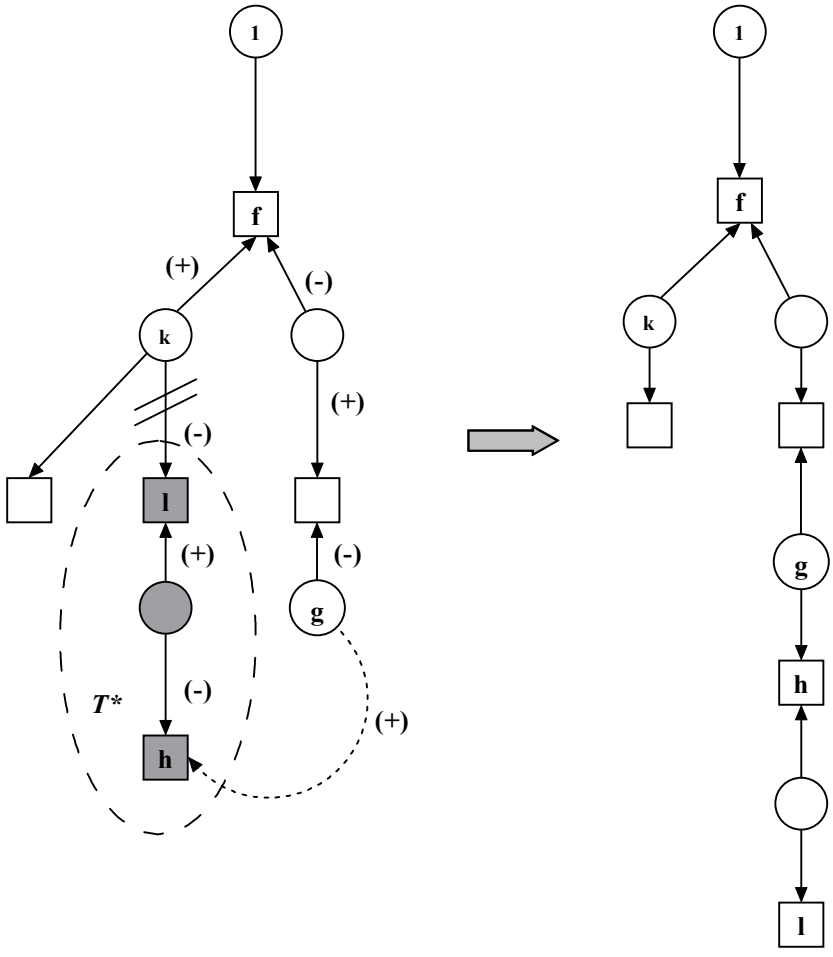
$$s = \begin{bmatrix}
0 & 0 & 0 & -40 & 5 & 60\\
20 & 9 & 0 & -3 & -46 & 26\\
17 & 28 & 0 & 0 & 0 & 84\\
-11 & -16 & -4 & -51 & 0 & 0
\end{bmatrix}$$

Έχοντας πλέον υπολογίσει όλα τα απαραίτητα στοιχεία μπορούμε να εφαρμόσουμε τον αλγόριθμο Simplex, η περιγραφή του οποίου αρχίζει παρακάτω.

2.1.2 Περιγραφή του Αλγορίθμου

Στη συνέχεια θα περιγράψουμε την εφαρμογή του πρωτεύοντος αλγορίθμου Simplex για την επίλυση του προβλήματος Μεταφοράς. Όπως είδαμε το αρχικό πρόβλημά μας περιγράφεται από ένα διμερές γράφημα με αποτέλεσμα τα τόξα των διαδοχικών δέντρων που σχηματίζονται να είναι εναλλάξ προς τα κάτω και προς τα πάνω, αναλόγως με το επίπεδο που βρίσκονται. Επίσης, όλα τα τόξα κατευθύνονται πάντα από κόμβους γραμμές σε κόμβους στήλες. Συνεπώς, αν πάρουμε δύο διαδοχικά τόξα, αποκλείεται αυτά να έχουν την ίδια φορά. Αυτή η παρατήρηση είναι πάρα πολύ σημαντική και ερμηνεύει όπως θα δούμε στη συνέχεια το ότι ένα πρόβλημα Μεταφοράς δεν είναι ποτέ απεριόριστο.

Ο αλγόριθμος ακολουθεί τη λογική του κλασσικού αλγορίθμου Simplex για το γενικό γραμμικό πρόβλημα. Σε κάθε επανάληψη, κάποια μεταβλητή, που ονομάζεται εισερχόμενο τόξο (g,h), εισέρχεται στη βάση ενώ κάποια άλλη μεταβλητή, που ονομάζεται εξερχόμενο τόξο (k,ℓ), βγαίνει από τη βάση. Αρχικά ο αλγόριθμος ελέγχει το πρόσημο των στοιχείων του πίνακα s. Αν είναι $s_{ij} \geq 0$ για όλα τα κελιά του πίνακα, τότε το τρέχον δέντρο είναι βέλτιστο και ο αλγόριθμος σταματά, αλλιώς προσδιορίζεται το εισερχόμενο τόξο (g,h). Παρακάτω βλέπουμε μια απεικόνιση μιας επανάληψης γενικής μορφής του αλγορίθμου Simplex.



Εικόνα 2.2 – Μια επανάληψη γενικής μορφής του αλγορίθμου Simplex

Το εισερχόμενο τόξο καθορίζεται από τον πίνακα των μειωμένων κοστών s και ισούται κάθε φορά με τους δείκτες ενός αρνητικού στοιχείου του πίνακα s, την τιμή του οποίου θέτουμε ίση με δ. Δηλαδή είναι $\delta = s_{gh}$. Στο παραπάνω σχήμα το εισερχόμενο τόξο ζωγραφίζεται με διακεκομμένες γραμμές. Το εισερχόμενο τόξο, αφού δεν είναι βασικό, θα

σχηματίζει έναν προσανατολισμένο κύκλο. Δημιουργούμε δύο σύνολα με τόξα, το C^+, το οποίο περιέχει το εισερχόμενο τόξο και όσα τόξα ανήκουν στον σχηματιζόμενο κύκλο και έχουν ίδια φορά με το εισερχόμενο τόξο και το C^-, το οποίο περιέχει τα υπόλοιπα τόξα, δηλαδή τα τόξα που ανήκουν στον κύκλο και έχουν αντίθετη φορά από αυτήν του εισερχομένου τόξου. Το εξερχόμενο τόξο (k,ℓ) θα είναι ένα τόξο του C^- το οποίο θα καθορίζεται από τη σχέση $x_{kl} = \varepsilon = \min\{x_{ij}(T) : (i,j) \in C^-\}$.Η ανανέωση του δέντρου γίνεται προσθέτοντας το εισερχόμενο τόξο και αφαιρώντας το εξερχόμενο. Έτσι προκύπτει ένα τελείως διαφορετικό δέντρο, όπως φαίνεται και στο σχήμα 2.2.

Το πιο ενδιαφέρον σημείο του αλγορίθμου αποτελεί η ανανέωση των τιμών $x_{ij}(T)$ και $s_{ij}(T)$, όπου T είναι το τρέχον δέντρο. Η ανανέωση των τιμών $x_{ij}(T)$ γίνεται με βάση τον τύπο:

$$x_{ij}(T') = \begin{cases} x_{ij}(T) + \varepsilon & ,\forall (i,j) \in C^+ \\ x_{ij}(T) - \varepsilon & ,\forall (i,j) \in C^- \end{cases}$$

,όπου $T' = T \cup (g,h) \sim (k,\ell)$ είναι το καινούριο δέντρο που προκύπτει κατά την επανάληψη του αλγορίθμου και $\varepsilon = x_{kl} = \min\{x_{ij}(T) : (i,j) \in C^-\}$. Η ανανέωση των μεταβλητών $s_{ij}(T)$ αποτελεί πιο δύσκολη διαδικασία. Βασικό στοιχείο στη διαδικασία ανανέωσης αποτελεί το αποκομμένο δέντρο $T*$, που είναι το υπόδεντρο του αρχικού δέντρου που προκύπτει κατά την αφαίρεση του εξερχόμενου τόξου (k,ℓ).Το δέντρο αυτό περιβάλλεται με ένα ελλειψοειδές στο σχήμα 2.2.

Η ανανέωση λοιπόν των τιμών $s_{ij}(T)$ πραγματοποιείται ως εξής: Για όλους τους κόμβους $b \in T*$ αν ο κόμβος b είναι κόμβος-γραμμή, προσθέτουμε(ή αφαιρούμε) την ποσότητα $\delta < 0$ στην γραμμή b του πίνακα s, αλλιώς (δηλαδή όταν ο κόμβος b είναι κόμβος-στήλη) αφαιρούμε(ή προσθέτουμε) την ποσότητα $\delta < 0$ στη στήλη b του πίνακα s. Για να αποφασίσουμε ποια μαθηματική πράξη θα εκτελεσθεί στις γραμμές και στις στήλες του πίνακα, πρέπει μετά το τέλος της ανανέωσης το μειωμένο κόστος του εισερχόμενου τόξου να γίνει ίση με το μηδέν, αφού αυτό γίνεται βασικό. Για να επιτευχθεί αυτό αλγοριθμικά, ελέγχουμε αν ο κόμβος h ανήκει στο δέντρο $T*$. Αν $h \in T*$ θέτουμε $q = -\delta > 0$, αλλιώς θέτουμε $q = \delta < 0$. Στη συνέχεια εάν ο κόμβος $b \in T*$ είναι κόμβος-γραμμή θα έχουμε $s_{\mathrm{b}j} := s_{\mathrm{b}j} - q$, $j = 1,...,n$ αλλιώς (δηλαδή ο κόμβος $b \in T*$ είναι κόμβος-στήλη) θα έχουμε $s_{i\,\mathrm{b}} := s_{i\,\mathrm{b}} + q$, $i = 1,...,m$.Μετά την ανανέωση όλων των μεταβλητών, έχει τελειώσει μια επανάληψη του αλγορίθμου και αναζητούμε νέο εισερχόμενο τόξο. Ακριβώς αυτή η μέθοδος ανανέωσης θα χρησιμοποιείται σε όλους τους αλγορίθμους που θα περιγράψουμε.

Τελειώνοντας με την περιγραφή του αλγορίθμου, θα αναφερθούμε σε μια πολύ σημαντική παρατήρηση που αφορά την επιλογή του εξερχόμενου τόξου. Είδαμε ότι το εξερχόμενο τόξο (k,ℓ) καθορίζεται από τον τύπο $x_{kl} = \varepsilon = \min\{x_{ij}(T) : (i,j) \in C^-\}$.Τι γίνεται όμως στην περίπτωση που έχουμε δύο ή περισσότερα τόξα που έχουν την ίδια ελάχιστη μεταβλητή απόφασης; Στην περίπτωση αυτή εφαρμόζουμε τον κανόνα κύκλωσης του *Cunninghum***[Cun]**, ο οποίος εγγυάται την κατασκευή μόνο ισχυρών δέντρων. Ισχυρό ονομάζεται το δέντρο εκείνο στο οποίο όλα τα εκφυλισμένα τόξα (i,j), δηλαδή τα τόξα με $x_{ij} = 0$, είναι προς τα κάτω. Για να εφαρμόσουμε τον κανόνα του *Cunninghum* θα πρέπει να αναφέρουμε τα παρακάτω:

- Ονομάζουμε επιτρεπτά τα τόξα $(r,t) \in C^-$ με $x_{rt} = \min\{x_{ij}(T) : (i,j) \in C^-\}$

- Ονομάζουμε κόμβο ένωση (*joint*) τον κόμβο που ανήκει στην τομή των δρόμων που διαγράφονται από τους κόμβους g και h προς τη ρίζα του δέντρου και έχει το μεγαλύτερο βάθος. Ο κόμβος ένωση στο σχήμα 2.2 είναι ο κόμβος f .

Ο κανόνας του *Cunninghum* επιλέγει το πρώτο επιτρεπτό τόξο που συναντάμε όταν διαγράφουμε τον σχηματιζόμενο κύκλο κατά τη φορά του εισερχόμενου τόξου, ξεκινώντας από τον κόμβο ένωση f .

2.1.3 Βηματική Περιγραφή του Αλγορίθμου

Στη συνέχεια θα παρουσιάσουμε τον αλγόριθμο σε μορφή βημάτων και βασισμένοι στα βήματα του αλγορίθμου θα λύσουμε ένα πρόβλημα.

ΒΗΜΑ 0:*(Καθορισμός αρχικών μεταβλητών)*
Υπολόγισε ένα εφικτό δέντρο ξεκινήματος T με τη μέθοδο της βορειοδυτικής γωνίας καθώς και τις δυϊκές μεταβλητές των κόμβων του δέντρου θέτοντας $u_1 = 0$ και $s_{ij}(T) = c_{ij} - u_i(T) - v_j(T) = 0$ για κάθε βασικό τόξο $(i, j) \in T$.

ΒΗΜΑ 1:*(Έλεγχος βελτιστότητας)*

αλλιώς πήγαινε στο βήμα 2.

ΒΗΜΑ 2:*(Επιλογή εισερχομένου τόξου)*
Υπολόγισε το εισερχόμενο τόξο (g, h) ελέγχοντας τα μειωμένα κόστη των μη βασικών τόξων με βάση τον τύπο $s_{gh} = \delta = \min\{s_{ij} : s_{ij} < 0\}$.

ΒΗΜΑ 3:*(Επιλογή εξερχόμενου τόξου)*
Υπολόγισε τα σύνολα τόξων C^+, C^- και καθόρισε τα επιτρεπτά τόξα (r, t) τέτοια ώστε να ισχύει $x_{rt} = \varepsilon = \min\{x_{ij} : (i, j) \in C^-\}$.Βρες τον κόμβο ένωση f .Τότε, το εξερχόμενο τόξο (k, ℓ) θα είναι το πρώτο τόξο που συναντάμε κατά τη διαγραφή του κύκλου με τη φορά του εισερχομένου τόξου ξεκινώντας από τον κόμβο ένωση f .*(Κανόνας ανακύκλωσης του Cunninghum)*.

ΒΗΜΑ 4:*(Ανανέωση των μεταβλητών)*

κάθε τόξο $(i, j) \in C^+$.Υπολόγισε το δέντρο T^* που είναι το δέντρο που αποκόβεται από τη ρίζα όταν αφαιρείται το εξερχόμενο τόξο. Αν $h \in T^*$ θέσε $q = -\delta > 0$, αλλιώς θέσε $q = \delta < 0$. Στη συνέχεια εάν ο κόμβος $b \in T^*$ είναι κόμβος-γραμμή θέσε $s_{bj}(T') = s_{bj}(T) - q$, $j = 1, ..., n$ αλλιώς (δηλαδή ο κόμβος $b \in T^*$ είναι κόμβος-στήλη) θέσε $s_{ib}(T') = s_{ib}(T) + q$, $i = 1, ..., m$.(Ανανεώνονται οι τιμές s_{ij} των τόξων (i, j) που προσπίπτουν σε κόμβους που ανήκουν στο δέντρο T^*)

ΒΗΜΑ 5:*(Ανανέωση του τρέχοντος δέντρου)*

Στη συνέχεια θα παρουσιάσουμε την ακριβή επίλυση ενός παραδείγματος. Θα χρησιμοποιήσουμε τα δεδομένα που χρησιμοποιήσαμε στο προηγούμενο παράδειγμα

(παράδειγμα 2.1) όπου εφαρμόσαμε τη μέθοδο της βορειοδυτικής γωνίας για την κατασκευή ενός εφικτού δέντρου ξεκινήματος.

Παράδειγμα 2.3

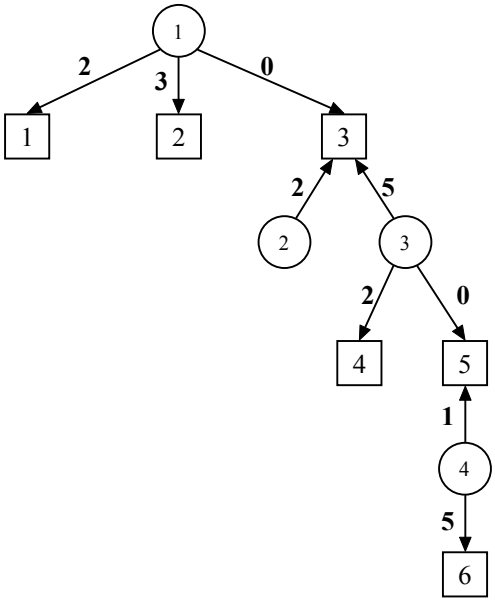
Δίνονται το διάνυσμα προσφοράς $a=\begin{bmatrix}5 & 2 & 7 & 6\end{bmatrix}$, ***το διάνυσμα ζήτησης*** $b=\begin{bmatrix}2 & 3 & 7 & 2 & 1 & 5\end{bmatrix}$, ***και ο πίνακας κόστους***

$$C=\begin{bmatrix}-5 & 49 & 28 & 20 & 47 & 43\\ 19 & 62 & 32 & 61 & 0 & 13\\ -2 & 63 & 14 & 46 & 28 & 53\\ 1 & 50 & 41 & 26 & 59 & 0\end{bmatrix}$$

για ένα 4×6 ***πρόβλημα Μεταφοράς. Να λυθεί με τον πρωτεύοντα αλγόριθμο Simplex που εφαρμόζει τον κανόνα περιστροφής του Cunninghum.***

Λύση:

Αρχικά, πρέπει να εφαρμόσουμε τη μέθοδο της βορειοδυτικής γωνίας για την κατασκευή του εφικτού δέντρου ξεκινήματος. Επίσης, πρέπει να υπολογίσουμε τις δυϊκές μεταβλητές και τον πίνακα μειωμένων κοστών s. Όλα αυτά έγιναν στο προηγούμενο παράδειγμα. Η λύση του προβλήματος θα περιγράφεται με ένα πίνακα στον οποίο θα περιέχουμε τα μειωμένα κόστη των τόξων του προβλήματος. Όπου υπάρχει βασικό τόξο, η τιμή της αντίστοιχης βασικής μεταβλητής θα βρίσκεται μέσα σε ένα γκρίζο κελί. Εννοείται ότι η αντίστοιχη τιμή του μειωμένου κόστους σε αυτό το κελί θα είναι ίση με το μηδέν. Συνεπώς το αρχικό δέντρο και ο αρχικός πίνακας που θα χρησιμοποιήσουμε φαίνονται στα παρακάτω σχήματα:



Εικόνα 2.3 – Το εφικτό δέντρο ξεκινήματος του προβλήματός μας.

Παρακάτω παρουσιάζουμε και τον αντίστοιχο πίνακα ξεκινήματος:

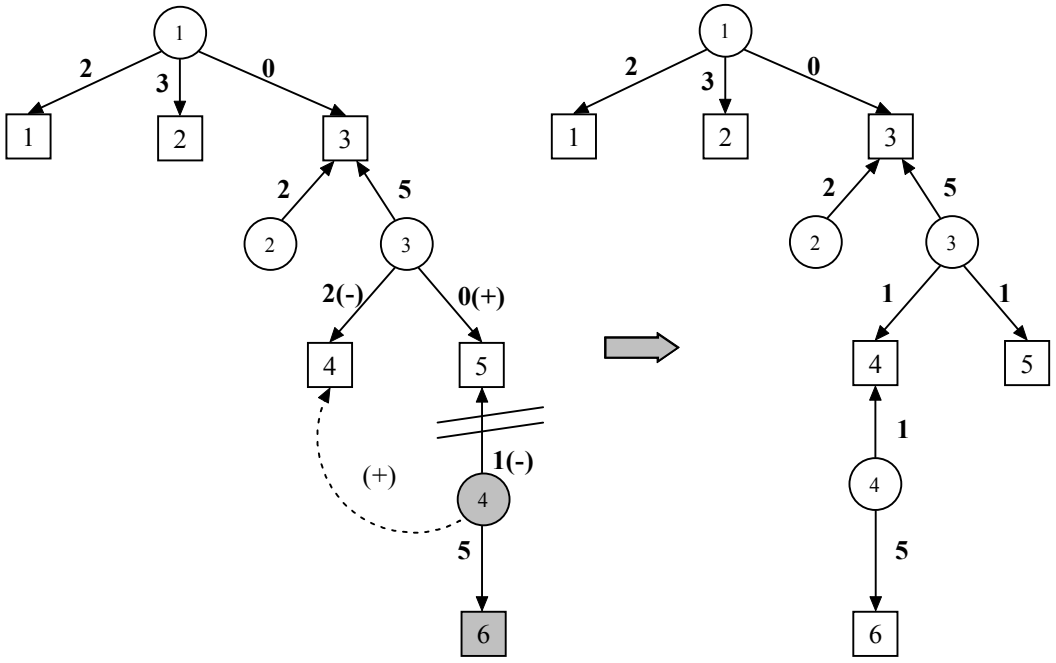
Πίνακας 2.13 – Ο πίνακας που αντιστοιχεί στο δέντρο του σχήματος 2.3

2	**3**	**0**	-40	5	60
20	9	**2**	-3	-46	26
17	28	**5**	**2**	**0**	84
-11	-16	-4	-51	**1**	**5**

Στη συνέχεια θα περιγράψουμε τις επαναλήψεις του αλγορίθμου για την επίλυση του προβλήματος:

Επανάληψη 1

- Είναι $s_{gh} = \delta = \min\{s_{ij} : s_{ij} < 0\} = s_{44} = -51$. Συνεπώς $\delta = -51$ και $(g,h) = (4,4)$
- Είναι $C^+ = \{(4,4),(3,5)\}$, $C^- = \{(3,4),(4,5)\}$.Συνεπώς για την επιλογή του εξερχόμενου τόξου θα έχουμε $x_{k\ell} = \varepsilon = \min\{x_{ij} : (i,j) \in C^-\} = \min\{x_{34}, x_{45}\} = 1 = x_{45}$. Άρα $\varepsilon = 1$ και $(k,\ell) = (4,5)$.
- Ανανεώνουμε τις τιμές x_{ij} ως εξής: Θέτουμε $x_{44} = 0+1 = 1$ και $x_{35} = 0+1 = 1$,αφού $(4,4),(3,5) \in C^+$ και $x_{34} = 2-1 = 1$, $x_{45} = 1-1 = 0$ αφού $(3,4),(4,5) \in C^-$.
- Η ανανέωση των μεταβλητών s_{ij} και του τρέχοντος δέντρου φαίνεται αναλυτικά στα παρακάτω σχήματα. Παρατηρείστε ότι οι κόμβοι του υπόδεντρου T^* ζωγραφίζονται με γκρι χρώμα.



2	**3**	**0**	-40	5	60	
20	9	**2**	-3	-46	26	
17	28	**5**	**2**	**0**	84	
-11	-16	-4	-51	**1**	**5**	**+51**

-51

2	**3**	**0**	-40	5	9
20	9	**2**	-3	-46	-25
17	28	**5**	**1**	**1**	33
40	35	47	**1**	51	**5**

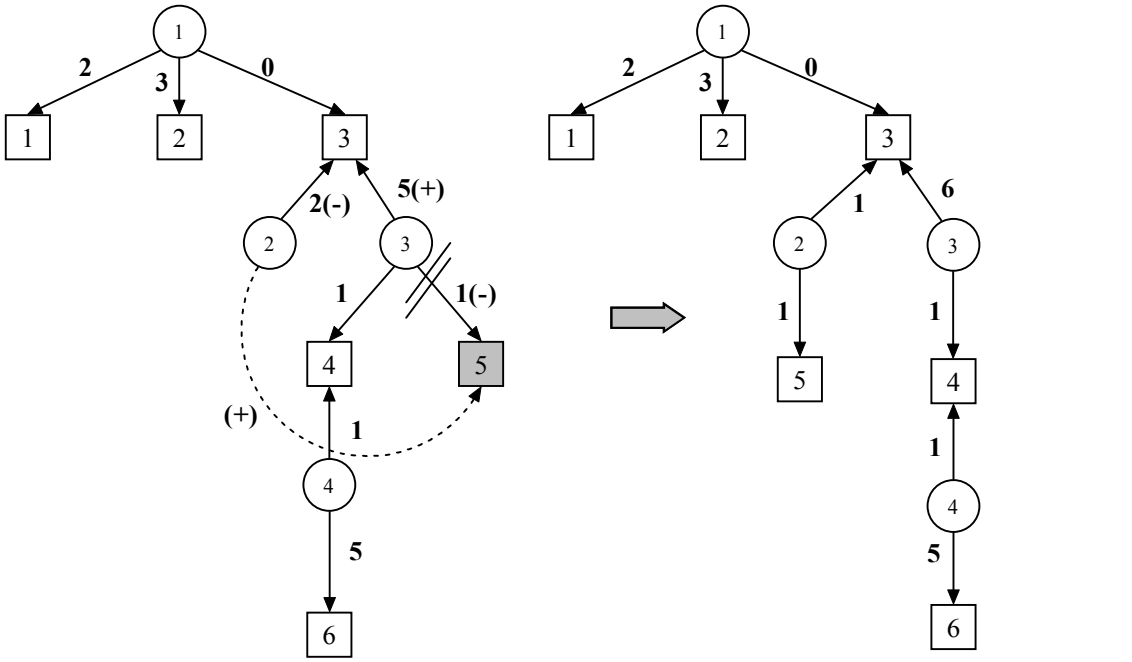
Εικόνα 2.4 - Η πρώτη επανάληψη του αλγορίθμου

Κάτω από κάθε δέντρο τοποθετούμε το πίνακα με τις τιμές s_{ij} και x_{ij} που αντιστοιχούν σε αυτά. Ειδικότερα, στον πρώτο πίνακα, δίπλα από κάθε γραμμή ή στήλη η τιμή της οποίας πρέπει να ανανεωθεί τοποθετούμε την τιμή ($+\delta$ ή $-\delta$) που προστίθεται κάθε φορά σε αυτήν. Στο συγκεκριμένο παράδειγμα επειδή μόνο ο κόμβος γραμμή 4 και ο κόμβος στήλη 6

ανήκουν στο υπόδεντρο $T*$, ανανεώνουμε μόνο τις τιμές s_{ij} στην τέταρτη γραμμή και έκτη στήλη του πίνακα.

Επανάληψη 2

- Είναι $s_{gh} = \delta = \min\{s_{ij} : s_{ij} < 0\} = s_{25} = -46$. Συνεπώς $\delta = -46$ και $(g,h) = (2,5)$
- Είναι $C^+ = \{(2,5),(3,3)\}, C^- = \{(3,5),(2,3)\}$.Συνεπώς για την επιλογή του εξερχόμενου τόξου θα έχουμε $x_{k\ell} = \varepsilon = \min\{x_{ij} : (i,j) \in C^-\} = \min\{x_{35}, x_{23}\} = 1 = x_{35}$. Άρα $\varepsilon = 1$ και $(k,\ell) = (3,5)$.
- Ανανεώνουμε τις τιμές x_{ij} ως εξής: Θέτουμε $x_{25} = 0+1 = 1$ και $x_{33} = 5+1 = 6$,αφού $(2,5),(3,3) \in C^+$ και $x_{23} = 2-1 = 1, x_{35} = 1-1 = 0$ αφού $(2,3),(3,5) \in C^-$.
- Η ανανέωση των μεταβλητών s_{ij} και του τρέχοντος δέντρου φαίνεται αναλυτικά στα παρακάτω σχήματα.



2	3	0	-40	5	9
20	9	2	-3	-46	-25
17	28	5	1	1	33
40	35	47	1	51	5

2	3	0	-40	51	9
20	9	1	-3	1	-25
17	28	6	1	46	33
40	35	47	1	97	5

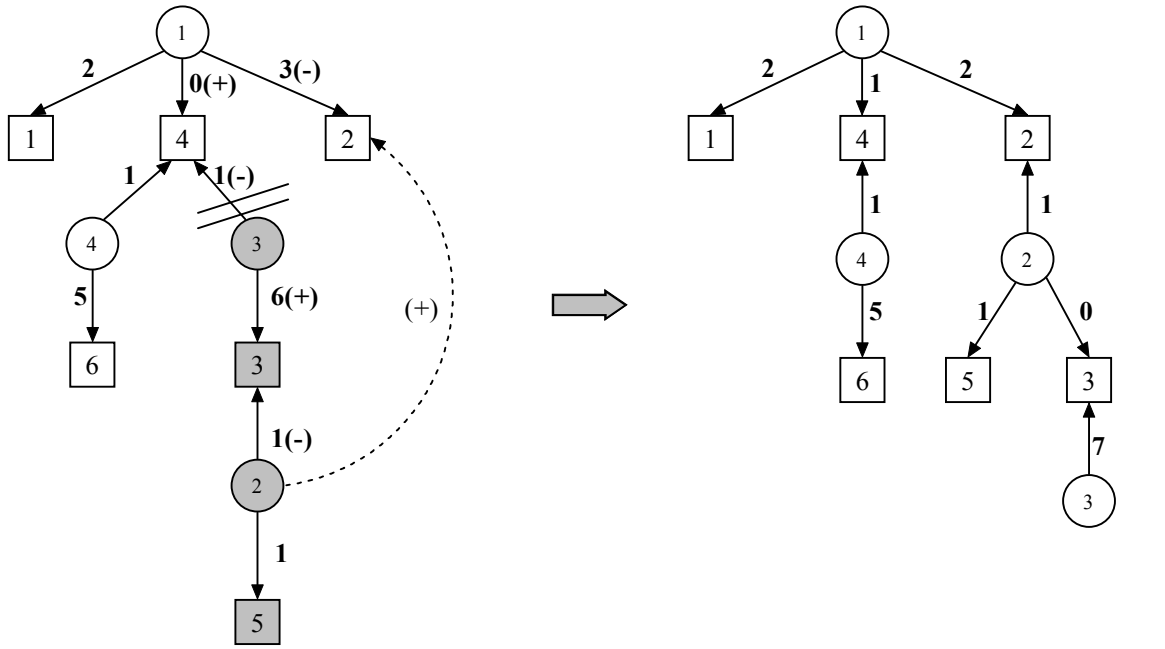
+46

Εικόνα 2.5 – Η δεύτερη επανάληψη του αλγορίθμου

Παρατηρούμε ότι αυτή τη φορά το δέντρο $T*$ αποτελείται μόνο από τον κόμβο στήλη 5. Συνεπώς προσθέτουμε την τιμή $-\delta$ μόνο στη στήλη 5 του πίνακα. Το ότι πρέπει να προσθέσουμε και όχι να αφαιρέσουμε $-\delta$ στη στήλη 5 του πίνακα, φαίνεται από το γεγονός ότι το μειωμένο κόστος του τόξου $(g,h) = (2,5)$ πρέπει να γίνει ίσο με το μηδέν. Αυτό όμως, μπορεί εύκολα να επαληθευτεί και αλγοριθμικά σύμφωνα με όσα αναφέρθηκαν στην περιγραφή του αλγορίθμου. Συνεπώς, επειδή $h \in T*$, θέτουμε $q = -\delta = 46 > 0$.Συνεπώς επειδή ο κόμβος $h \in T*$ είναι κόμβος στήλη, θα έχουμε $s_{ih} = s_{ih} + 46,\ i = 1,...,4$. Αυτή τη μέθοδο όπως θα δούμε παρακάτω θα χρησιμοποιήσουμε και για τον προγραμματισμό της συνάρτησης που θα ανανεώνει τον πίνακα s.

Επανάληψη 3

- Είναι $s_{gh} = \delta = \min\{s_{ij} : s_{ij} < 0\} = s_{14} = -40$. Συνεπώς $\delta = -40$ και $(g,h) = (1,4)$
- Είναι $C^+ = \{(1,4),(3,3)\}, C^- = \{(3,4),(1,3)\}$.Συνεπώς για την επιλογή του εξερχόμενου τόξου θα έχουμε $x_{k\ell} = \varepsilon = \min\{x_{ij} : (i,j) \in C^-\} = \min\{x_{34}, x_{13}\} = 0 = x_{13}$. Άρα $\varepsilon = 0$ και $(k,\ell) = (1,3)$.
- Επειδή $\varepsilon = 0$, η επανάληψη αυτή δε θα επηρεάσει τις τιμές των μεταβλητών απόφασης x_{ij}. Απλώς θέτουμε την τιμή της μεταβλητής απόφασης του εισερχομένου τόξου $x_{14} = 0$.Η επανάληψη αυτή, δηλαδή όπου $\varepsilon = 0$, ονομάζεται εκφυλισμένη, αφού δεν μεταβάλλει την τιμή της αντικειμενικής συνάρτησης.
- Η ανανέωση των μεταβλητών s_{ij} και του τρέχοντος δέντρου φαίνεται αναλυτικά στα παρακάτω σχήματα.



2	**3**	40	**0**	91	49	
-20	-31	**1**	-3	**1**	-25	**+31**
-23	-12	**6**	**1**	46	33	**+31**
0	-5	47	**1**	97	**5**	
		-31		**-31**		

2	**1**	9	**1**	60	49
11	**1**	**0**	28	**1**	6
8	19	**7**	31	46	64
0	-5	16	**1**	66	**5**

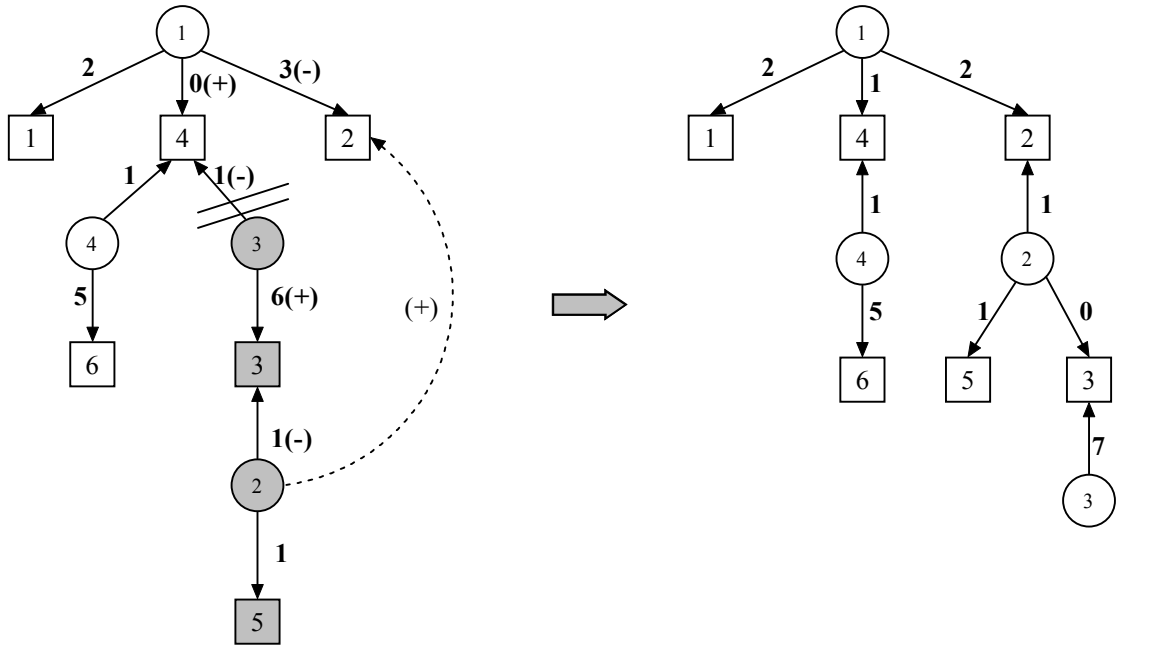
Εικόνα 2.6 – Η τρίτη επανάληψη του αλγορίθμου

Στην παραπάνω επανάληψη βλέπουμε ότι το υπόδεντρο $T*$ περιέχει περισσότερους κόμβους. Η ανανέωση όλων των τιμών φαίνεται αναλυτικά στους παραπάνω πίνακες. Αν θελήσουμε να υπολογίσουμε την τιμή της αντικειμενικής συνάρτησης για να δούμε κατά πόσο έχει βελτιωθεί μέχρι τώρα θα έχουμε $z(T_3) = \sum_{(i,j)\in T_3} c_{ij}x_{ij} = 325$. Η τιμή της αντικειμενικής συνάρτησης για το αρχικό εφικτό δέντρο T_1 είναι $z(T_1) = \sum_{(i,j)\in T_1} c_{ij}x_{ij} = 422$.

Συνεπώς μέχρι στιγμής έχουμε μια βελτίωση κατά $z(T_1) - z(T_3)$, η οποία όμως δεν είναι βέλτιστή.

Επανάληψη 4

- Είναι $s_{gh} = \delta = \min\{s_{ij} : s_{ij} < 0\} = s_{22} = -31$. Συνεπώς $\delta = -31$ και $(g,h) = (2,2)$
- Είναι $C^+ = \{(2,2),(1,4),(3,3)\}$, $C^- = \{(2,3),(3,4),(1,2)\}$. Εδώ παρατηρούμε ότι $\varepsilon = \min\{x_{ij} : (i,j) \in C^-\} = \min\{x_{12}, x_{34}, x_{23}\} = 1 = x_{23} = x_{34}$. Άρα έχουμε δύο επιτρεπτά τόξα τα $(2,3)$ και $(3,4)$.Ο κόμβος ένωση είναι η ρίζα του δέντρου, επειδή είναι ο κόμβος που ανήκει στην τομή των συνόλων $P[T,g], P[T,h]$ και έχει το μικρότερο βάθος, όπου $P[T,w]$ είναι το σύνολο των κόμβων που βρίσκονται στο μονοπάτι από έναν κόμβο w προς τη ρίζα του δέντρου. Άρα, στο συγκεκριμένο παράδειγμα ο κόμβος ένωση είναι ο κόμβος 1. Συνεπώς το εξερχόμενο τόξο είναι το $(k,\ell) = (3,4)$ επειδή είναι το πρώτο τόξο που συναντάμε όταν διαγράφουμε τον κύκλο κατά τη φορά του εισερχομένου τόξου.
- Ανανεώνουμε τις τιμές x_{ij} ως εξής: Θέτουμε $x_{22} = 0+1 = 1$, $x_{14} = 0+1 = 1$ και $x_{33} = 6+1 = 7$,αφού $(2,2),(1,4),(3,3) \in C^+$ και $x_{12} = 3-1 = 2$, $x_{34} = 1-1 = 0$ και $x_{23} = 1-1 = 0$ αφού $(1,2),(3,4),(2,3) \in C^-$.
- Η ανανέωση των μεταβλητών s_{ij} και του τρέχοντος δέντρου φαίνεται αναλυτικά στα παρακάτω σχήματα.



2	3	40	0	91	49	
-20	-31	1	-3	1	-25	+31
-23	-12	6	1	46	33	+31
0	-5	47	1	97	5	
		-31		-31		

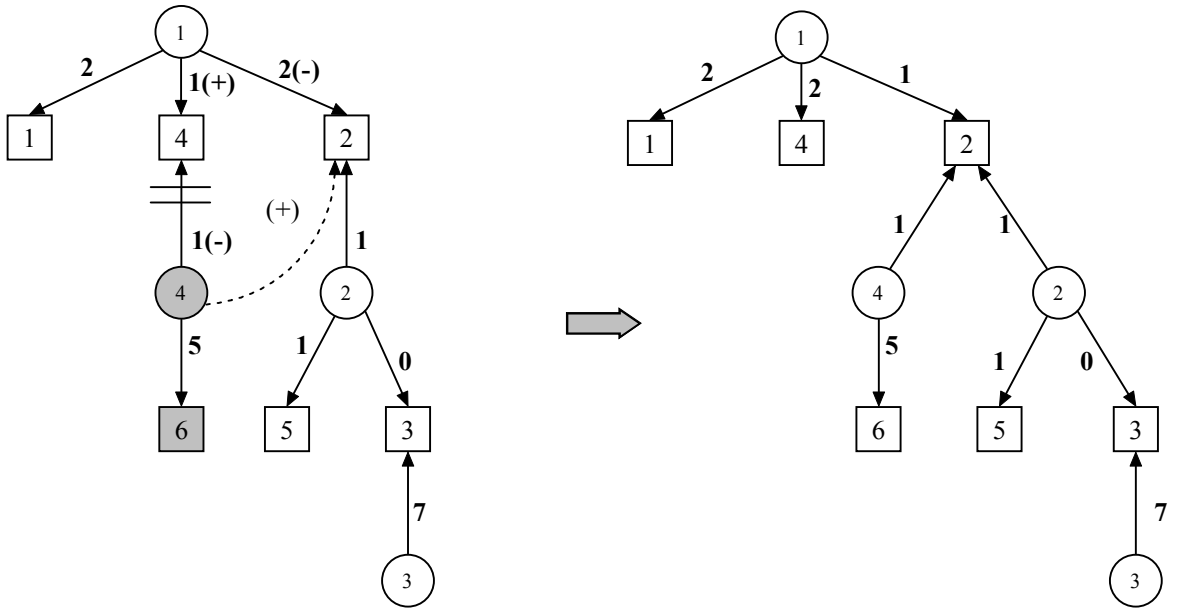
2	1	9	1	60	49
11	1	0	28	1	6
8	19	7	31	46	64
0	-5	16	1	66	5

Εικόνα 2.7 – Η τέταρτη επανάληψη του αλγορίθμου

Παρατηρούμε ότι ο αλγόριθμος δεν έχει τελειώσει ακόμα αφού υπάρχουν ακόμα αρνητικές τιμές s_{ij} , πράγμα που σημαίνει ότι το τρέχον δέντρο δεν είναι βέλτιστο. Έτσι, προχωράμε στην επανάληψη 5.

Επανάληψη 5

- Είναι $s_{gh} = \delta = \min\{s_{ij} : s_{ij} < 0\} = s_{42} = -5$. Συνεπώς $\delta = -5$ και $(g,h) = (4,2)$
- Είναι $C^+ = \{(4,2),(1,4)\}$, $C^- = \{(1,2),(4,4)\}$.Συνεπώς για την επιλογή του εξερχόμενου τόξου θα έχουμε $x_{k\ell} = \varepsilon = \min\{x_{ij} : (i,j) \in C^-\} = \min\{x_{12}, x_{44}\} = 1 = x_{44}$. Άρα $\varepsilon = 1$ και $(k,\ell) = (4,4)$.
- Ανανεώνουμε τις τιμές x_{ij} ως εξής: Θέτουμε $x_{42} = 0+1 = 1$ και $x_{14} = 1+1 = 2$,αφού $(4,2),(1,4) \in C^+$ και $x_{12} = 2-1 = 1, x_{44} = 1-1 = 0$ αφού $(1,2),(4,4) \in C^-$.
- Η ανανέωση των μεταβλητών s_{ij} και του τρέχοντος δέντρου φαίνεται αναλυτικά στα παρακάτω σχήματα.



2	**1**	9	**1**	60	49	
11	**1**	**0**	28	**1**	6	
8	19	**7**	31	46	64	
0	-5	16	**1**	66	**5**	+5

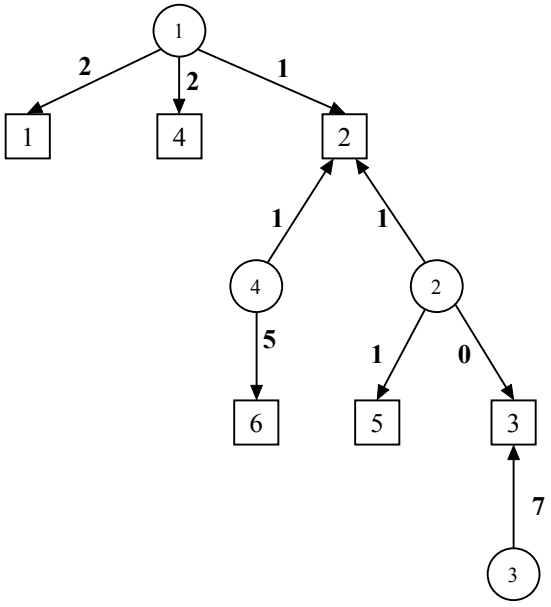
-5

2	**1**	9	**1**	60	44
11	**1**	**0**	28	**1**	1
8	19	**7**	31	46	59
5	**1**	21	5	71	**5**

Εικόνα 2.8 – Η πέμπτη επανάληψη του αλγορίθμου και το τελικό βέλτιστο δέντρο

Παρατηρούμε ότι ο πίνακας s περιέχει μόνο θετικές τιμές, άρα ο αλγόριθμος τερματίζει. Το βέλτιστο δέντρο T έχει προσδιοριστεί μετά από 5 επαναλήψεις και είναι αυτό του σχήματος 2.9. Η βέλτιστη τιμή της αντικειμενικής συνάρτησης θα είναι $z(T) = \sum_{(i,j)\in T} c_{ij}x_{ij} = 289$, η οποία είναι μικρότερη από την αρχική τιμή της αντικειμενικής συνάρτησης(422).

Στη συνέχεια θα παρουσιάσουμε τον ψευδοκώδικα του αλγορίθμου σε μορφή κλήσεως συναρτήσεων και κατόπιν ορισμένες αποτελεσματικές τεχνικές προγραμματισμού του αλγορίθμου που στοχεύουν σε καλές υπολογιστικές επιδόσεις.



Εικόνα 2.9 – Το βέλτιστο δέντρο του προβλήματος

2.1.4 Προγραμματισμός του Αλγορίθμου

Πριν προχωρήσουμε στην παρουσίαση των σημαντικότερων συναρτήσεων που χρησιμοποιήσαμε για τον προγραμματισμό του αλγορίθμου, θα παρουσιάσουμε τις σημαντικότερες δομές και μεταβλητές που χρησιμοποιήσαμε για την συγγραφή των συναρτήσεων.

Έτσι λοιπόν, για τον προγραμματισμό του αλγορίθμου χρησιμοποιήσαμε για την αποθήκευση του δέντρου το διάνυσμα πατέρα – κόμβου **`p()`** και το διάνυσμα βάθους **`d()`** όπως αυτά παρουσιάσθηκαν στο εισαγωγικό κεφάλαιο. Επίσης χρησιμοποιήσαμε τα διανύσματα **`u()`** και **`v()`** για την αποθήκευση των δυϊκών μεταβλητών και το δισδιάστατο πίνακα **`s`** για την αποθήκευση των μειωμένων κοστών s_{ij}.

Επίσης, για να υλοποιούνται με σχετική υπολογιστική ευκολία όλες οι λειτουργίες του αλγορίθμου Simplex χρησιμοποιήσαμε ακόμη δύο δομές δεδομένων, το διάνυσμα **`t()`**, το οποίο χρειάζεται για τον προσδιορισμό των συνόλων C^+ και C^-, και το διάνυσμα **`xv()`**, στο οποίο αποθηκεύονται οι τιμές των μεταβλητών απόφασης των τόξων(που αποθηκεύονται επίσης και στο δισδιάστατο πίνακα **`x`**).Το διάνυσμα **`t()`** ορίζεται ως εξής:

$$t(i) = \begin{cases} 0\ , & (p(i), i) \in T \\ 1\ , & (i, p(i)) \in T \end{cases}$$

Το διάνυσμα **`xv()`** αποτελεί μια έξυπνη μέθοδος αποθήκευσης των μεταβλητών x_{ij}. Σε κάθε θέση του i αποθηκεύεται η τιμή της μεταβλητής απόφασης του τόξου $(i, p(i))$ ή $(p(i), i)$ αναλόγως με την τιμή του διανύσματος **`t()`** στη συγκεκριμένη θέση. Επίσης χρησιμοποιείται και μια μεταβλητή **`simplex`** που παίζει το ρόλο ενός flag που καθορίζει τον τερματισμό ή μη του αλγορίθμου.

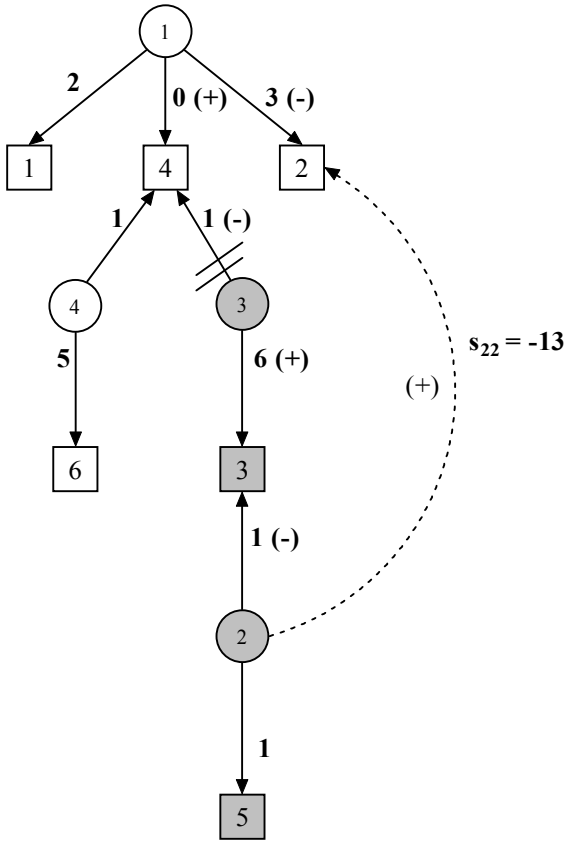
Παράλληλα ο αλγόριθμος χρησιμοποιεί τις θέσεις μνήμης **`min`**,**`g`**,**`h`** και για την αποθήκευση των τρεχόντων τιμών της μεταβλητής δ και των δύο άκρων του εισερχομένου τόξου αντίστοιχα, τις θέσεις μνήμης **`e`**,**`k`**,**`l`**,**`Cplus()`**,**`Cminus()`** για την πρόσκαιρη αποθήκευση των μεταβλητών ε, των δύο άκρων του εξερχόμενου τόξου και των τόξων που περιέχονται στα σύνολα C^+ και C^- αντίστοιχα. Στις μεταβλητές **`Cplus()`**,**`Cminus()`**

αποθηκεύονται τα τόξα των κύκλων ως εξής: Αν `Cplus(i)== k` τότε στο σύνολο C^+ περιέχεται ή το τόξο $(k, p(k))$ ή το τόξο $(p(k), k)$ ανάλογα με την τιμή του διανύσματος t στη συγκεκριμένη θέση. Εδώ βλέπουμε πόσο σημαντική είναι η δομή t. Μας δείχνει κατά κάποιο τρόπο τη «φύση» του δέντρου.

Τέλος, από τις πλέον σημαντικότερες μεταβλητές που χρησιμοποιεί ο αλγόριθμος 2.1 είναι οι μεταβλητές `Tcut(),e1,f1,e2,f2,z()`. Στην μεταβλητή `Tcut()` ο αλγόριθμος αποθηκεύει τους κόμβους που περιέχονται στο υπόδεντρο T^*. Επίσης, θα δηλώνουμε με `e1` το άκρο του εισερχόμενου τόξου (g,h) και με `f1` το άκρο του εξερχόμενου τόξου (k,ℓ) τα οποία ανήκουν στο υπόδεντρο T^*.Αναλόγως, θα συμβολίζουμε με `e2` το άκρο του εισερχόμενου τόξου (g,h) και με `f2` το άκρο του εξερχόμενου τόξου (k,ℓ) τα οποία δεν ανήκουν στο υπόδεντρο T^*. Τότε, ο δρόμος του δέντρου T^* που ενώνει τον κόμβο $e1$ με τον κόμβο $f1$ ονομάζεται στέλεχος (*steam*) και δηλώνεται στον αλγόριθμο με τη μεταβλητή `z[]`. Το πρώτο στοιχείο του διανύσματος z είναι πάντα ο κόμβος $e1$ και το τελευταίο ο κόμβος $f1$.

Η χρησιμοποίηση των παραπάνω μεταβλητών και δομών δεδομένων για τον προγραμματισμό του αλγορίθμου Simplex δεν αποτελεί μοναδική μέθοδο προγραμματισμού αλγορίθμων τέτοιας μορφής. Μας προσφέρουν όμως μεγάλη ευκολία στην γραφική αναπαράσταση δέντρων καθώς και στην ανανέωση των μεταβλητών που χρησιμοποιεί ο αλγόριθμος. Παράλληλα, παρέχουν στον προγραμματιστή την δυνατότητα να γράψει κώδικα με πολύ καλές υπολογιστικές επιδόσεις.

Για να κατανοηθεί από τον αναγνώστη ποιες είναι οι δομές και οι μεταβλητές που



Εικόνα 2.10

υπολογίζει ο αλγόριθμος που προγραμματίσαμε, θα παρουσιάζουμε την «κατάσταση» της μνήμης για την παραπάνω επανάληψη του αλγορίθμου(εικόνα 2.10).Συνεπώς θα έχουμε:

- $p = [-1 \quad 7 \quad 8 \quad 8 \quad 1 \quad 1 \quad 3 \quad 1 \quad 2 \quad 4]$.Παρατηρείστε ότι ο k κόμβος – στήλη αποθηκεύεται στο διάνυσμα ως $k+m$, όπου m το πλήθος των κόμβων γραμμών του προβλήματος, έτσι ώστε οι κόμβοι στήλες να είναι διακεκριμένοι.
- $d = [0 \quad 4 \quad 2 \quad 2 \quad 1 \quad 1 \quad 3 \quad 5 \quad 3 \quad 4]$.Παρατηρείστε ότι οι κόμβοι γραμμές βρίσκονται σε άρτια επίπεδα ενώ οι κόμβοι στήλες σε περιττά επίπεδα.
- $t = [-1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1]$.
- $X = \begin{bmatrix} 2 & 3 & \infty & 0 & \infty & \infty \\ \infty & \infty & 1 & \infty & 1 & \infty \\ \infty & \infty & 6 & 1 & \infty & \infty \\ \infty & \infty & \infty & 1 & \infty & 5 \end{bmatrix}$.
- $xv = [0 \quad 1 \quad 1 \quad 1 \quad 2 \quad 3 \quad 6 \quad 0 \quad 1 \quad 5]$.
- $\min = -13$, $g = 2$, $h = 2$.Παρατηρείστε ότι στις περιπτώσεις αποθήκευσης άκρων εισερχομένου ή εξερχόμενου τόξου δε χρησιμοποιούμε τη μέθοδο αποθήκευσης που εφαρμόστηκε στο διάνυσμα p αλλά τα αποθηκεύουμε με την απόλυτη τιμή τους.
- $e = 1$, $k = 3$, $\ell = 4$.
- $Tcut = [3 \quad 7 \quad 2 \quad 9]$.
- $e1 = 2$, $f1 = 3$, $e2 = 6$, $f2 = 8$, $z = [2 \quad 7 \quad 3]$.

Στη συνέχεια παρουσιάζεται το κυρίως πρόγραμμα που αποτελείται από τις συναρτήσεις που γράψαμε για την υλοποίηση του αλγορίθμου. Τα ονόματα των μεταβλητών είναι τα ίδια με αυτά που παρουσιάσθηκαν παραπάνω.

Αλγόριθμος T1

Ο παρακάτω αλγόριθμος λύνει το ισοζυγισμένο πρόβλημα μεταφοράς, χρησιμοποιώντας τον πρωτεύοντα αλγόριθμο Simplex και την μέθοδο της βορειοδυτικής γωνίας για την κατασκευή ενός εφικτού δέντρου.

Είσοδος: *Το διάνυσμα προσφοράς* **A(mx1)**,*το διάνυσμα ζήτησης* **B(nx1)** *και ο πίνακας κόστους* **C(mxn)**.

Έξοδος : *Η λύση του προβλήματος Μεταφοράς* **X(mxn)**.

```
1.  [X,S]=NorthWestCorner(A,B);
2.  [U,V]=DualVariables(S,C,m,n);
3.  S=ReducedCosts(m,n,C,U,V);
4.  p=InitializeNodeFather(X,m,n);
5.  d=InitializeNodeDepth(p,m,n);
6.  t=InitializeNodeDirection(m,n);
7.  xv=Vectorx(p,t,X,m,n);
8.  min=-inf;
9.  simplex=1;
10. while simplex==1
11.   [min,g,h]=EnteringArc(S,m,n);
12.   if min~=0
13.         simplex=1;
14.         [e,k,l,Cplus,Cminus]=LeavingArc(g,h,p,m,n,d,t,xv);
15.         [e1,e2,f1,z,Tcut]=Steam(p,g,h,k,l,d,m,n);
16.         [S,X]=UpdateVariables(min,e,g,h,k,l,Tcut,Cminus,Cplus,m,n,t,p,S,X);
17.         d=UpdateNodeDepth(p,m,n,Tcut,z,d,e1,e2,f1);
18.         p=UpdateNodeFather(p,z,e2,f1);
19.         xv=Vectorx(p,t,X,m,n);
20.   else
21.         simplex=0;
22.   end
23. end
```

Αλγόριθμος 2.1.1 – Ο κώδικας υλοποίησης του πρωτεύοντος αλγορίθμου Simplex για το πρόβλημα Μεταφοράς

Η γλώσσα που χρησιμοποιήθηκε είναι το Matlab, η οποία δεν περιορίζει τον αναγνώστη να υλοποιήσει τον αλγόριθμο σε μια άλλη δομημένη γλώσσα, λόγω της έλλειψης ιδιαίτερων προγραμματιστικών χαρακτηριστικών.

Παρακάτω, θα παρουσιάσουμε τις υλοποιήσεις και αναλύσεις των πιο σημαντικών κατά την άποψη μας συναρτήσεων. Όλη η υλοποίηση του αλγορίθμου μπορεί να βρεθεί στο παράρτημα της εργασίας.

Οι επιμέρους συναρτήσεις – αλγόριθμοι του αλγορίθμου 2.1.1 που κρίνουμε σκόπιμο να παρουσιάσουμε είναι οι ακόλουθες:

- **[X , S] = NorthWestCorner(A , B)**

Αλγόριθμος NorthWestCorner
Ο αλγόριθμος που προσδιορίζει του εφικτό δέντρο ξεκινήματος με τη μέθοδο της βορειοδυτικής γωνίας.
Είσοδος: *Το διάνυσμα προσφοράς* **A(mx1)** *και το διάνυσμα ζήτησης* **B(nx1)**.
Έξοδος : *Το εφικτό δέντρο* **X(mxn)** *και ο αντίστοιχος πίνακας μειωμένων κοστών* **S(mxn)** .

```
1.  function [X,S]=NorthWestCorner(A,B)
2.  m=length(A);
3.  n=length(B);
4.  for i=1:m
5.    for j=1:n
6.      X(i,j)=inf;
7.      S(i,j)=inf;
8.    end
9.  end
10. i=1;
11. j=1;
12. while (i<=m)&(j<=n)
13.   t=min(A(i),B(j));
14.   X(i,j)=t;
15.   S(i,j)=0;
16.   A(i)=A(i)-X(i,j);
17.   B(j)=B(j)-X(i,j);
18.   if (A(i)==0)&(B(j)>0)
19.     i=i+1;
20.   elseif (B(j)==0)&(A(i)>0)
21.     j=j+1;
22.   elseif ((A(i)==0)&(B(j)==0))|(A(i)==B(j))
23.     if (j~=n)
24.       X(i,j+1)=A(i);
25.       S(i,j+1)=0;
26.     end
27.    i=i+1;
28.    j=j+1;
29.   end
30. end
```

Αλγόριθμος 2.1.2 – Ο αλγόριθμος υπολογισμού του δέντρου ξεκινήματος με τη μέθοδο της βορειοδυτικής γωνίας.

Ο παραπάνω κώδικας ακολουθεί ακριβώς τη λογική που παρουσιάσαμε κατά την περιγραφή του εφικτού δέντρου της βορειοδυτικής γωνίας. Αρχικά, θέτει όλα τα στοιχεία των πινάκων **X** και **S** ίσα με άπειρο.(γραμμές 4-9) Στη συνέχεια(γραμμές 12 -30), χρησιμοποιείται ένας βρόχος του **while** και υπολογίζεται το εφικτό δέντρο με τον αλγόριθμο που παρουσιάσθηκε. Εδώ θα πρέπει να κάνουμε μια πολύ σημαντική παρατήρηση. Θα μπορούσαμε για την αρχικοποίηση των πινάκων **X** και **S** να χρησιμοποιήσουμε τις απευθείας εντολές ανάθεσης του Matlab, χωρίς να χρησιμοποιήσουμε δύο βρόχους του **for**, πράγμα που θα απλούστευε τον κώδικά μας. Σκοπός μας όμως είναι να προγραμματίσουμε τους αλγορίθμους σε μια γλώσσα που είναι όσο το δυνατότερο συμβατή με τις κλασσικές γλώσσες δομημένου προγραμματισμού, όπως είναι η Pascal και η C. Δηλαδή, ο αναγνώστης αν θελήσει να

προγραμματίσει τους αλγορίθμους αυτούς σε κάποια άλλη γλώσσα προγραμματισμού όπως οι προαναφερθέντες, να μπορεί να βασιστεί στον κώδικά μας.

- **[U , V]=DualVariables(S , C , m , n)**

Αλγόριθμος DualVariables
Ο αλγόριθμος που υπολογίζει τα διανύσματα των δυϊκών μεταβλητών του προβλήματος.
Είσοδος: S(mxn), C(mxn), *οι διαστάσεις του προβλήματος* **m, n**
Έξοδος : *Τα διανύσματα των δυϊκών μεταβλητών* **U(m), V(n)**

```
1.  function [U,V]=DualVariables(S,C,m,n)
2.  for i=1:m
3.    U(i)=inf;
4.  end
5.  for j=1:n
6.    V(j)=inf;
7.  end
8.  U(1)=0;
9.  done1=0;
10. done2=0;
11. while (done1<m)|(done2<n)
12.   done1=0;
13.   done2=0;
14.   for i=1:m
15.     for j=1:n
16.       if S(i,j)==0
17.         if (U(i)~=inf)&(V(j)==inf)
18.           V(j)=C(i,j)-U(i);
19.         elseif (V(j)~=inf)&(U(i)==inf)
20.           U(i)=C(i,j)-V(j);
21.         end
22.       end
23.     end
24.   end
25.   for i=1:m
26.     if (U(i)~=inf)
27.       done1=done1+1;
28.     end
29.   end
30.   for j=1:n
31.     if(V(j)~=inf)
32.       done2=done2+1;
33.     end
34.   end
35. end
```

Αλγόριθμος 2.1.3 – Ο αλγόριθμος υπολογισμού των διανυσμάτων των δυϊκών μεταβλητών.

Ο παραπάνω αλγόριθμος υπολογίζει όπως προαναφέρθηκε τα διανύσματα των δυικών μεταβλητών U και V. Τα διανύσματα αυτά θα υπολογιστούν μόνο μία φορά και θα χρησιμοποιηθούν για τον υπολογισμό των μειωμένων κοστών. Κατά τη διάρκεια εκτέλεσης του αλγορίθμου θα ανανεώνονται μόνο τα μειωμένα κόστη s_{ij} χωρίς να γίνεται η ανανέωση των δυϊκών μεταβλητών του προβλήματος.

Ο αλγόριθμος 2.1.3 αποτελεί μια αρκετά καλή μέθοδο για τον υπολογισμό των δυϊκών μεταβλητών. Λύνει το γραμμικό σύστημα που προκύπτει κάθε φορά χωρίς να χρησιμοποιεί κάποια ενσωματωμένη συνάρτηση του Matlab για το λόγο που αναφέρθηκε προηγουμένως. Αρχικά απειρίζει τα διανύσματα (γραμμές 2-7) και στη συνέχεια για κάθε τόξο $(i, j) \in T$, δηλαδή για κάθε τόξο (i, j) όπου $s_{ij} = 0$, αν έχει υπολογιστεί τουλάχιστον μια μεταβλητή

εκ των U_i ή V_j, υπολογίζει την εναπομείνασα V_j ή U_i, χρησιμοποιώντας τη σχέση $s_{ij} = 0$ (γραμμές 11-24). Η διαδικασία έχει ως βάση το γεγονός ότι θέτουμε $U_1 = 0$ (γραμμή 8) και στη συνέχεια δουλεύει κατά κάποιο τρόπο «αναδρομικά». Ο αλγόριθμος τερματίζει και υπολογίζει τις δυϊκές μεταβλητές όταν $U_i < \infty$, $\forall\, i = 1..m$ και $V_j < \infty$,$\forall\, j = 1..n$ (γραμμές 25-34).

- **x = Preorder(root, p, m, n)**

Αλγόριθμος Preorder
Ο αλγόριθμος που υπολογίζει την προδιατεταγμένη διάσχιση ενός ριζωμένου δέντρου.
Είσοδος: *Η ρίζα του δέντρου* **root,** *το διάνυσμα πατέρα-κόμβου* **p(m+n), m, n**
Έξοδος : *Το διάνυσμα της προδιατεταγμένης διάσχισης* **x(m+n)**

```
1.  function x=Preorder(root,p,m,n)
2.  global dd
3.  global y
4.  y=1;
5.  x=[];
6.  c=pr(root,p,m,n);
7.  x=dd;
8.  dd=[];
9.   function c=pr(root,p,m,n)
10. global dd
11. global y
12. c=root;
13. dd(y)=c;
14. y=y+1;
15. for i=1:m+n
16.   if p(i)==root
17.     c=pr(i,p,m,n);
18.   end
19. end
```

Αλγόριθμος 2.1.4 – Ο αλγόριθμος υπολογισμού του διανύσματος προδιατεταγμένης διάσχισης.

Ο παραπάνω αλγόριθμος υλοποιεί μια διαδικασία που καλείται μέσα στον αλγόριθμο 2.1.1 από άλλες διαδικασίες. Παράγει το διάνυσμα της προδιατεταγμένης διάσχισης ενός δέντρου. Είναι μια αναδρομική συνάρτηση που καλείται αρχικά για τη ρίζα του δέντρου, στη συνέχεια για τα παιδιά της ρίζας(γραμμή 16) και αναλόγως η διαδικασία συνεχίζεται μέχρι να εξετασθούν όλοι οι κόμβοι του δέντρου.

Στη συγκεκριμένη περίπτωση, χρησιμοποιήσαμε ορισμένα ιδιαίτερα προγραμματιστικά χαρακτηριστικά του Matlab που δεν προσφέρονται σε γλώσσες προγραμματισμού όπως η Pascal και η C, έτσι ώστε όχι μόνο να προγραμματίσουμε τον αλγόριθμο αναδρομικά αλλά και στο τέλος να λαμβάνουμε ως αποτέλεσμα ολόκληρο το διάνυσμα που περιέχει τις επιστρεφόμενες τιμές των αναδρομικών κλήσεων που εκτελέσθηκαν. Δηλαδή, με τη μέθοδο που χρησιμοποιήσαμε, μπορέσαμε και αποθηκεύσαμε στο τελικό διάνυσμα που επιστρέφει η συνάρτηση ότι εξαγόταν από τη στοίβα κατά τις αναδρομικές κλήσεις της συνάρτησης. Αυτό επιτεύχθηκε με την ειδική χρήση των μεταβλητών που δηλώνονται μέσα στον κώδικα ως **`global`**. Κάτι τέτοιο σε μια άλλη γλώσσα προγραμματισμού θα ήταν πολύ δύσκολο έως αδύνατο, αφού κατά τη χρησιμοποίηση αναδρομικών συναρτήσεων επιστρέφεται η τιμή που «πετάει» η στοίβα της μνήμης του υπολογιστή κατά την τελευταία αναδρομική κλήση. Παραδείγματος χάριν, ο αναδρομικός προγραμματισμός σε γλώσσα C της συνάρτησης της υψώσεως σε δύναμη ($a^n = a \cdot a^{n-1}$) έτσι ώστε με την έξοδο από τη συνάρτηση να παράγεται ένα διάνυσμα που να περιέχει τις τιμές

που υπολογίστηκαν και εξάχθηκαν από τη στοίβα κατά τη διάρκεια εκτέλεσης του προγράμματος (δηλαδή για το 2^5 να παράγεται το διάνυσμα $w=[1 \quad 2 \quad 4 \quad 8 \quad 16 \quad 32]$) είναι κάτι πολύ δύσκολο και απαιτεί επέμβαση του προγραμματιστή στον τρόπο διαχείρισης της μνήμης. Σε Pascal δε, είναι μάλλον αδύνατο.

Θα μπορούσαμε βέβαια να προγραμματίσουμε τον αλγόριθμο χωρίς αναδρομή, υλοποιώντας μια στοίβα. Αυτό έγινε κατά την πρώτη φάση αναπτύξεως του λογισμικού και διαπιστώθηκε ότι για μεγάλα προβλήματα ο χρόνος εκτέλεσης του αλγορίθμου καταναλωνόταν κατά το ήμισυ στην κλήση της συγκεκριμένης συνάρτησης, πράγμα που επέβαλε τον αναδρομικό προγραμματισμό(που θεωρητικά αποδεικνύεται αποδοτικότερος) της συνάρτησης, μιας και μιλάμε για αποτελεσματικό προγραμματισμό αλγορίθμων.

- **[e1, e2, f1, z, Tcut] = Steam(p , g , h , k , l , d , m , n)**

Αλγόριθμος Steam
Ο αλγόριθμος που υπολογίζει όλες τις μεταβλητές που σχετίζονται με το δέντρο T.*
Είσοδος: p(m+n),*το εισερχόμενο τόξο* **(g,h),** *το εξερχόμενο τόξο* **(k,l),** *το διάνυσμα βάθους* **d(m+n)**, **m, n** .
Έξοδος : *Οι μεταβλητές* **e1**, **e2, f1**, *το στέλεχος* **z( )** *και το αποκομμένο δέντρο* **Tcut**.

```
1.  function [e1,e2,f1,z,Tcut]=Steam(p,g,h,k,l,d,m,n)
2.  z=0;
3.  if d(k)>d(l+m)
4.    f1=k;
5.    f2=l+m;
6.  else
7.    f1=l+m;
8.    f2=k;
9.  end
10. Tcut=Preorder(f1,p,m,n);
11. for i=1:length(Tcut)
12.   if g==Tcut(i)
13.     e1=g;
14.     e2=h+m;
15.   else
16.     if h+m==Tcut(i)
17.       e1=h+m;
18.       e2=g;
19.     end
20.   end
21. end
22. z(1)=e1;
23. i=1;
24. while z(i)~=f1
25.   z(i+1)=p(z(i));
26.   i=i+1;
27. end
```

Αλγόριθμος 2.1.5 – Ο αλγόριθμος Steam.

Στον παραπάνω αλγόριθμο βλέπουμε ότι γίνονται υπολογισμοί που σχετίζονται με το αποκομμένο δέντρο T^*.Έχει προηγηθεί ο υπολογισμός του εισερχόμενου και εισερχόμενου τόξου μέσω των συναρτήσεων **`EnteringArc`**(γραμμή 11 του αλγορίθμου 2.1.1) και **`LeavingArc`**(γραμμή 14 του αλγορίθμου 2.1.1), οπότε μας είναι διαθέσιμες οι μεταβλητές g,h και k,ℓ .Η ανάλυση των συναρτήσεων αυτών για το συγκεκριμένο αλγόριθμο δε κρίθηκε σκόπιμη.

Ας προχωρήσουμε τώρα στην ανάλυση του κώδικα του αλγορίθμου 2.1.5. Αρχικά υπολογίζονται οι μεταβλητές $f1, f2$ (γραμμές 3-9), οι μεταβλητές $e1, e2$ όπως αυτές ορίσθηκαν παραπάνω.(γραμμές 11-21) Επίσης ο παραπάνω κώδικας υπολογίζει και το αποκομμένο δέντρο T^*, το οποίο έχει πάντα ρίζα τον κόμβο $f1$ και έτσι χρησιμοποιούμε τη

συνάρτηση **`Preorder()`** με όρισμα για τη ρίζα τον κόμβο $f1$ για να υπολογίζουμε τους κόμβους που ανήκουν στο δέντρο $T*$(γραμμή 10). Τέλος, υπολογίζουμε και το διάνυσμα του στελέχους z. Αυτό επιτυγχάνεται ως εξής: Θέτουμε αρχικά $z(1)=e1$, αφού όπως είπαμε ο κόμβος $e1$ αποτελεί πάντα τον πρώτο κόμβο του στελέχους. Οι υπόλοιποι κόμβοι του στελέχους θα είναι όπως είδαμε όλοι εκείνοι οι κόμβοι που βρίσκονται στο μονοπάτι από τον κόμβο $e1$ μέχρι τον κόμβο $f1$(γραμμές 24 -27). Το διάνυσμα του στελέχους είναι μια πάρα πολύ σημαντική δομή δεδομένων αφού όπως θα δούμε χρησιμεύει στην ανανέωση σημαντικών δομών του αλγορίθμου.

- **[S, X] =UpdateVariables(min ,e ,g ,h ,k ,l ,Tcut ,Cminus ,Cplus ,m ,n ,t ,p ,S ,X)**

Αλγόριθμος UpdateVariables
Ο αλγόριθμος που ανανεώνει τις μεταβλητές X και S του αλγορίθμου.
Είσοδος: min, e, g, h, k, l, Tcut(), Cminus(), Cplus(), m, n, t(m+n), p(m+n), S(mxn), X(mxn) .
Έξοδος :*Οι ανανεωμένες μήτρες* **S(mxn), X(mxn)** .

```
1.  function [S,X]=UpdateVariables(min,e,g,h,k,l,Tcut,Cminus,Cplus,m,n,t,p,S,X)
2.  i=1;
3.  while Cminus(i)~=0
4.    if (t(Cminus(i))==1)&(t(Cminus(i))~=-1)
5.      X(Cminus(i),p(Cminus(i))-m)=X(Cminus(i),p(Cminus(i))-m)-e;
6.    elseif t(Cminus(i))==0
7.      X(p(Cminus(i)),Cminus(i)-m)=X(p(Cminus(i)),Cminus(i)-m)-e;
8.    end
9.    i=i+1;
10. end
11. i=1;
12. while Cplus(i)~=0
13.   if (t(Cplus(i))==1)&(t(Cplus(i))~=-1)
14.     X(Cplus(i),p(Cplus(i))-m)=X(Cplus(i),p(Cplus(i))-m)+e;
15.   elseif t(Cplus(i))==0
16.     X(p(Cplus(i)),Cplus(i)-m)=X(p(Cplus(i)),Cplus(i)-m)+e;
17.   end
18.   i=i+1;
19. end
20. X(k,l)=inf;
21. X(g,h)=e;
22. for i=1:length(Tcut)
23.   if h+m==Tcut(i)
24.     min=-min;
25.   end
26. end
27. for i=1:length(Tcut)
28.   if Tcut(i)<=m
29.     for j=1:n
30.       S(Tcut(i),j)=S(Tcut(i),j)-min;
31.     end
32.   else
33.     for j=1:m
34.       S(j,Tcut(i)-m)=S(j,Tcut(i)-m)+min;
35.     end
36.   end
37. end
```

Αλγόριθμος 2.1.6 – Ο αλγόριθμος ανανέωσης των μητρών S,X.

Ο παραπάνω αλγόριθμος ανανεώνει τις βασικότερες μεταβλητές του αλγορίθμου, δηλαδή τις μήτρες S, X .Αρχικά, ανανεώνει τη μήτρα X χρησιμοποιώντας τα διανύσματα **`Cminus`** και **`Cplus`** που αποθηκεύουν πληροφορίες για τα σύνολα C^- και C^+ αντίστοιχα. Συνεπώς, αναλόγως το σύνολο που ανήκει κάθε τόξο του κύκλου, αφαιρούμε ή προσθέτουμε τη

μεταβλητή **e**. Αυτό επιτυγχάνεται στις γραμμές 2-21 του αλγορίθμου 2.1.6.Στις επόμενες γραμμές του κώδικα(22-37) ανανεώνονται οι τιμές s_{ij} των τόξων $(i,j) \in T*$ με τον τρόπο που περιγράψαμε στην αρχή της παρουσίασης του αλγορίθμου.

- **p =UpdateNodeFather(p,z,e2,f1)**

Αλγόριθμος UpdateNodeFather
Ο αλγόριθμος που ανανεώνει το διάνυσμα πατέρα - κόμβου.
Είσοδος: p(m+n), **z(), e2, f1**.
Έξοδος :*Το ανανεωμένο διάνυσμα πατέρα- κόμβου* **p(m+n)**.

```
1. function p=UpdateNodeFather(p,z,e2,f1)
2. p(z(1))=e2;
3. i=1;
4. while z(i)~=f1
5.   p(z(i+1))=z(i);
6.   i=i+1;
7. end
```

Αλγόριθμος 2.1.7 – Ο αλγόριθμος ανανέωσης του διανύσματος πατέρα – κόμβου p.

Στον παραπάνω αλγόριθμο φαίνεται καθαρά ότι ανανεώνονται μόνο οι τιμές του διανύσματος p που αντιστοιχούν σε κόμβους που περιέχονται στο στέλεχος. Αυτή η παρατήρηση είναι πάρα πολύ σημαντική και φανερώνει τη μεγάλη βοήθεια που μας παρέχει το διάνυσμα του στελέχους z.Αν ο αλγόριθμος δε χρησιμοποιούσε το διάνυσμα του στελέχους θα έπρεπε να υπολογίζει εξαρχής το διάνυσμα πατέρα – κόμβου καλώντας κάθε φορά την συνάρτηση **InitializeNodeFather**(γραμμή 4 του αλγορίθμου 2.1.1) με διαφορετική κάθε φορά μεταβλητή εισόδου X.Αυτό θα είχε ως αποτέλεσμα να αυξηθεί ο χρόνος εκτέλεσης του αλγορίθμου ιδιαίτερα όταν τρέχουμε μεγάλα προβλήματα, πράγμα που το διαπιστώσαμε τρέχοντας μεγάλα προβλήματα και εφαρμόζοντας και τις δύο μεθόδους.

- **d = UpdateNodeDepth(p,m,n,Tcut,z,d,e1,e2,f1)**

Ίσως η σημαντικότερη προσφορά του διανύσματος του στελέχους z βρίσκεται στη χρησιμοποίησή του από τη συνάρτηση ανανέωσης του διανύσματος που αποθηκεύει τα βάθη των κόμβων. Στη συνέχεια θα παρουσιάσουμε έναν αποτελεσματικό αλγόριθμο ανανέωσης των βαθών των κόμβων του δέντρου. Πριν παραθέσουμε τον κώδικα του αλγορίθμου θα παρουσιάσουμε την μέθοδο που ακολουθούμε κάθε φορά για την ανανέωση του διανύσματος βάθους.

Έστω λοιπόν d το διάνυσμα βαθών που αντιστοιχεί σε ένα δέντρο T και d' το διάνυσμα που αντιστοιχεί στο ανανεωμένο δέντρο T'. Θα παρουσιάσουμε μια αποτελεσματική διαδικασία παραγωγής του διανύσματος d', δίνοντας πρώτα ορισμένους αναγκαίους ορισμούς.

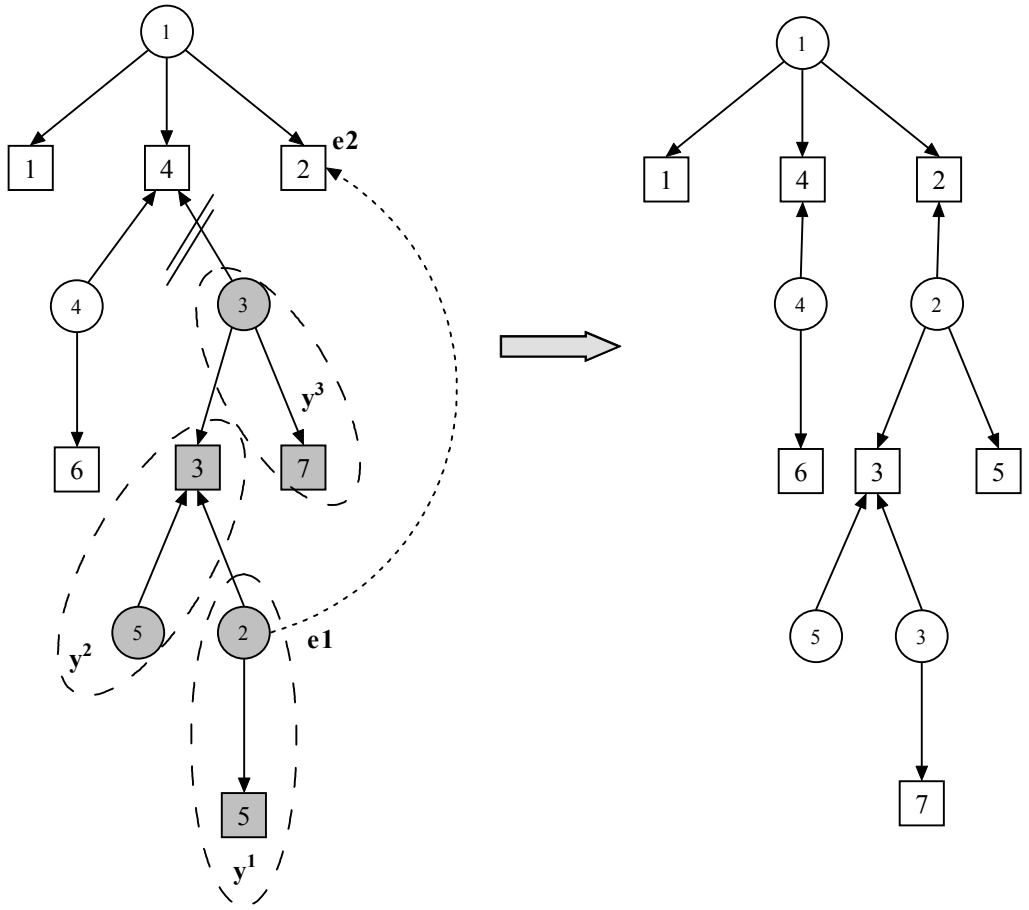
Έστω $T* \subseteq T$ το γνωστό σε εμάς δέντρο που αποκόπτεται κατά την αφαίρεση του εξερχόμενου τόξου (k,ℓ) και y η προδιατεταγμένη διάσχιση αυτού του δέντρου. Έχουμε αναφέρει επίσης ότι πάντα οι κόμβοι του στελέχους z θα ανήκουν στο δέντρο $T*$.Ορίζουμε τα διανύσματα $y^i \subseteq y$, $i = 1,...,w$, όπου w το μήκος του διανύσματος του στελέχους z ως εξής:

$$y^i = \begin{cases} f(z_i),\ i = 1 \\ f(z_i) \sim (f(z_i) \cap f(z_{i-1})),\ 2 \le i \le w \end{cases}$$

,όπου $f : N \to N^{1\times j}$ είναι μια συνάρτηση που επιστρέφει το διάνυσμα $f(x)$ της προδιατεταγμένης διάσχισης ενός δέντρου ρίζας x .Παρατηρείστε ότι η μεταβλητή y^i είναι διάνυσμα ενώ η μεταβλητή z_i είναι στοιχείο διανύσματος, δηλαδή βαθμωτό μέγεθος.

Από τον παραπάνω ορισμό συμπεραίνουμε ότι ένα διάνυσμα y^i περιέχει τους κόμβους εκείνους που ανήκουν στην προδιατεταγμένη διάσχιση ενός δέντρου ρίζας z_i αλλά δεν ανήκουν στην προδιατεταγμένη διάσχιση του υπόδεντρου ρίζας z_{i-1}.

Ας θεωρήσουμε τώρα μια γενική επανάληψη του αλγορίθμου όπου από ένα δέντρο T προκύπτει το δέντρο T'. Μια τέτοια επανάληψη φαίνεται στο παρακάτω σχήμα:



Εικόνα 2.11 – Μια γενική επανάληψη του αλγορίθμου όπου φαίνονται οι δομές y^1, y^2, y^3.

Παρατηρείστε τις δομές $y^i, i = 1,2,3$.Αν προσέξουμε τα βάθη των κόμβων $i \in y^1$ στο δέντρο T και στο δέντρο T', παρατηρούμε ότι υπάρχει ένας ακέραιος αριθμός r τέτοιος ώστε

$$d'(i) = d(i) + r \quad \forall i \in y^i$$

Έτσι, αν προσέξουμε και τις θέσεις των κόμβων $e1, e2$ μπορούμε να αποδείξουμε ότι είναι:

$$r = d(e_2) - d(e_1) + 1$$

Στην εικόνα 2.11 θα είναι συνεπώς $y^1 = [2 \quad 10]$, $y^2 = [5 \quad 8]$, $y^3 = [3 \quad 12]$ και $r = 1 - 4 + 1 = -2$. Για κάθε κόμβο $x \in y_i$ μπορεί επίσης να αποδειχθεί ότι το νέο βάθος του $d'(x)$ θα δίνεται από τον τύπο:

$$d'(x) = d(x) + r + 2(i-1), \quad x \in y^i$$

Εύκολα μπορούμε να καταλάβουμε ότι οι κόμβοι που δεν ανήκουν στο δέντρο T^* δεν μεταβάλουν το βάθος τους. Συνεπώς, μπορούμε να προγραμματίσουμε έναν αλγόριθμο που να ανανεώνει μόνο τα βάθη των κόμβων που ανήκουν στο υπόδεντρο T^* με τον τρόπο που περιγράψαμε. Είναι αξιοσημείωτο ότι η εφαρμογή της προαναφερθείσας μεθόδου σε μεγάλα προβλήματα μεταφοράς έριξε το χρόνο εκτέλεσης του αλγορίθμου στο μισό. Ο αλγόριθμος αυτός είναι ο παρακάτω:

Αλγόριθμος UpdateNodeDepth
Ο αλγόριθμος που ανανεώνει το διάνυσμα βάθους.
Είσοδος: p(m+n), m, n,Tcut(), z(), d(m+n), e1, e2, f1
Έξοδος : *Το ανανεωμένο διάνυσμα βάθους* **d(m+n)**.

```
1.  function d=UpdateNodeDepth(p,m,n,Tcut,z,d,e1,e2,f1)
2.  r=d(e2)-d(e1)+1;
3.  Tdep=Tcut;
4.  pdep=p;
5.  for i=1:length(z)
6.    ye=Preorder(z(i),pdep,m,n);
7.    for j=1:length(ye)
8.      if (any(ye(j)==Tdep))
9.        d(ye(j))=d(ye(j))+r+2*(i-1);
10.     end
11.   end
12.   for ee=1:length(Tdep)
13.     if any(Tdep(ee)==ye)
14.       pdep(Tdep(ee))=-7;
15.       Tdep(ee)=0;
16.     end
17.   end
18. end
```

Αλγόριθμος 2.1.8 – Ο αλγόριθμος ανανέωσης του διανύσματος βαθών d.

Ο παραπάνω αλγόριθμος υπολογίζει τις προδιατάξεις για κάθε κόμβο του στελέχους και μετά αφαιρεί αυτούς τους κόμβους από μια δομή `Tdep` η οποία αρχικοποιείται με τους κόμβους που περιέχονται στο δέντρο T^*, με αποτέλεσμα κάθε φορά να γίνονται οι συγκρίσεις με τους σωστούς κόμβους.

Στο σημείο αυτό τελειώνει η παρουσίαση του αλγορίθμου Simplex για το πρόβλημα Μεταφοράς. Παρουσιάσθηκαν οι πιο σημαντικές συναρτήσεις και κυρίως αυτές που αποτέλεσαν και τη βάση για τον προγραμματισμό και των άλλων αλγορίθμων. Είναι ευνόητο πως αν οι συναρτήσεις αυτές χρησιμοποιηθούν στους επόμενους αλγορίθμους δε θα αναλυθούν ξανά. Θα αναλύονται μόνο εκείνες οι συναρτήσεις που υλοποιούν τις ιδιαιτερότητες του συγκεκριμένου αλγορίθμου.

Ακολουθεί η περιγραφή του πρωτεύοντος αλγορίθμου Simplex για το πρόβλημα Αντιστοίχησης.

2.2 ΕΦΑΡΜΟΓΗ ΣΤΟ ΠΡΟΒΛΗΜΑ ΑΝΤΙΣΤΟΙΧΗΣΗΣ

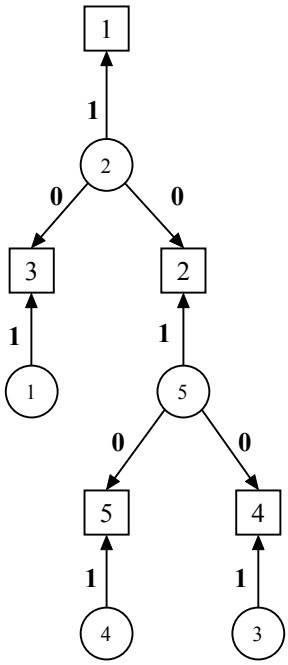
Στη συνέχεια θα περιγράψουμε πως μπορεί ο πρωτεύων αλγόριθμος Simplex να εφαρμοστεί για τη επίλυση του προβλήματος αντιστοίχησης. Θα παραθέσουμε την περιγραφή του αλγορίθμου, τη βηματική του εκτέλεση σε μορφή ψευδοκώδικα καθώς και λεπτομέρειες για έναν αποτελεσματικό τρόπο προγραμματισμού του αλγορίθμου.

2.2.1 Υπολογισμός ενός Εφικτού Δέντρου Ξεκινήματος

Όπως αναφέρθηκε και στην εισαγωγή, το πρόβλημα Αντιστοίχησης είναι μια ειδική περίπτωση του προβλήματος Μεταφοράς από το οποίο και προκύπτει θέτοντας $m = n$ και $a_i = b_i = 1,\ i = 1,...,n$. Λόγω της ειδικής δομής του προβλήματος οι μεταβλητές απόφασης x_{ij} θα ανήκουν στο διάστημα $\{0,1\}$. Συγκεκριμένα, κάθε εφικτό δέντρο του προβλήματος αντιστοίχησης περιέχει n μεταβλητές απόφασης x_{ij} ίσες με 1 και $n-1$ μεταβλητές ίσες με 0. Σε αυτή την παράγραφο, θα περιγράψουμε τον υπολογισμό ενός εφικτού δέντρου ξεκινήματος για την επίλυση του προβλήματος Αντιστοίχησης χρησιμοποιώντας τον πρωτεύοντα αλγόριθμο Simplex.Το εφικτό δέντρο ξεκινήματος που θα χρησιμοποιήσουμε θα έχει ρίζα τον κόμβο στήλη 1. Επειδή όλα τα εκφυλισμένα τόξα είναι προς τα κάτω, το δέντρο δεν μπορεί να περιέχει φύλλα που είναι κόμβοι στήλες, εκτός βέβαια από τη ρίζα του δέντρου. Δουλεύοντας επαγωγικά αποδεικνύεται ότι αν ένα τόξο (i, j) είναι προς τα πάνω, δηλαδή αν κατευθύνεται από τα φύλλα προς τη ρίζα του δέντρου, θα είναι $x_{ij} = 1$ αλλιώς θα είναι $x_{ij} = 0$.Έτσι συμπεραίνουμε ότι κάθε κόμβος στήλη εκτός της ρίζας θα έχει βαθμό 2 στο ισχυρό δέντρο ξεκινήματος εκτός της ρίζας η οποία θα έχει βαθμό 1.Συνοψίζοντας όλα τα παραπάνω, μπορούμε να πούμε ότι γενικά ένα ισχυρό δέντρο για προβλήματα αντιστοίχησης ριζωμένο σε έναν κόμβο στήλη θα έχει τις παρακάτω ιδιότητες:

- Όλα τα $n-1$ εκφυλισμένα τόξα είναι προς τα κάτω
- Για όλα τα n προς τα πάνω τόξα (i, j) είναι $x_{ij} = 1$.
- Όλα τα φύλλα του δέντρου(εκτός της ρίζας) είναι κόμβοι γραμμές.
- Κάθε κόμβος στήλη εκτός της ρίζας έχει βαθμό 2.

Στο παρακάτω σχήμα βλέπουμε ένα ισχυρό δέντρο για ένα πρόβλημα αντιστοίχησης:



Εικόνα 2.12 - Ένα ισχυρό δέντρο για ένα πρόβλημα Αντιστοίχησης

Το εφικτό δέντρο ξεκινήματος που θα χρησιμοποιήσουμε για την επίλυση του προβλήματος αντιστοίχησης είναι ειδικής δομής. Συγκεκριμένα, όταν έχουμε να λύσουμε ένα πρόβλημα Αντιστοίχησης $n \times n$, το εφικτό δέντρο ξεκινήματος θα είναι ένας δρόμος*(path)*που έχει ρίζα από τον κόμβο στήλη 1 προς τον κόμβο γραμμή n. Το δέντρο αυτό προκύπτει εφαρμόζοντας τη μέθοδο της βορειοδυτικής γωνίας όπως αυτήν χρησιμοποιήθηκε για την κατασκευή ενός εφικτού δέντρου ξεκινήματος στο πρόβλημα Μεταφοράς. Είναι δηλαδή πάντα ο δρόμος κόμβος στήλη 1, κόμβος γραμμή 1, κόμβος στήλη 2, κόμβος γραμμή 2,…., κόμβος στήλη n, κόμβος γραμμή n.

Εικόνα 2.13 – Το εφικτό δέντρο ξεκινήματος για ένα πρόβλημα Αντιστοίχησης *nxn.*

Οι αντίστοιχες δυϊκές μεταβλητές των κόμβων u_i και v_j, $i, j = 1,...,n$ υπολογίζονται όπως είδαμε και στο πρόβλημα Μεταφοράς. Αρχικά θέτουμε $v_1 = 0$, αφού ρίζα του δέντρου είναι τώρα ο κόμβος στήλη 1. Στη συνέχεια, λύνουμε το παρακάτω γραμμικό σύστημα εξισώσεων:

$$s_{ii} = c_{ii} - u_i - v_i = 0,\ i = 1,...,n$$
$$s_{ij+1} = c_{ii+1} - u_i - v_{i+1} = 0,\ i = 1,...,n-1$$

, αφού τα βασικά τόξα του δέντρου ξεκινήματος είναι της μορφής *(i,i), i=1,...,n* και $(i, i+1)$,$i = 1,...,n-1$. Αφού υπολογίσω τις δυϊκές μεταβλητές που αντιστοιχούν στους κόμβους του δέντρου της εικόνας 2.13, τότε υπολογίζω τα μειωμένα κόστη s_{ij} για κάθε $(i, j) \notin T$, όπου T το εφικτό δέντρο ξεκινήματος, θέτοντας $s_{ij} = c_{ij} - u_i - v_j$. Στη συνέχεια

θα παραθέσουμε ένα παράδειγμα υπολογισμού ενός εφικτού δέντρου ξεκινήματος και των αντίστοιχων δυϊκών μεταβλητών.

Παράδειγμα 2.4

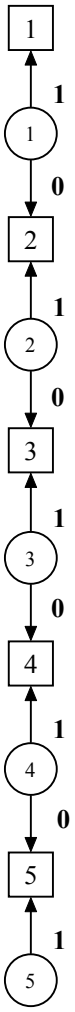
Δίνεται ο πίνακας κόστους

$$C = \begin{bmatrix} 3 & -3 & 19 & 10 & -27 \\ 0 & -1 & 42 & 46 & 77 \\ 70 & 32 & 43 & 71 & 90 \\ 34 & 1 & 0 & 0 & 8 \\ 1 & 7 & 34 & 65 & -7 \end{bmatrix}.$$

Να προσδιορισθεί το εφικτό δέντρο ξεκινήματος για ένα 5×5 ***πρόβλημα Αντιστοίχησης και να προσδιορισθεί ο πίνακας των μειωμένων κοστών.***

Λύση

Το εφικτό δέντρο ξεκινήματος είναι της ίδιας δομής για κάθε πρόβλημα Αντιστοίχησης και υπολογίζεται χωρίς τη χρησιμοποίηση των δεδομένων του παραδείγματος. Συνεπώς, το εφικτό δέντρο ξεκινήματος θα είναι το παρακάτω:



Εικόνα 2.14 – Το εφικτό δέντρο ξεκινήματος για ένα 5x5 πρόβλημα Αντιστοίχησης

Στη συνέχεια θα υπολογίσουμε τις δυϊκές μεταβλητές u_i και v_j, $i, j = 1,\ldots,n$ του προβλήματος θέτοντας $v_1 = 0$ και $s_{ij} = 0$ για κάθε τόξο $(i, j) \in T$. Η μέθοδος που

ακολουθείται είναι ακριβώς η ίδια που εφαρμόστηκε και στο πρόβλημα Μεταφοράς. Συνεπώς θα πρέπει να επιλύσουμε το παρακάτω σύστημα εξισώσεων:

$$\left.\begin{aligned} v_1 &= 0 \\ c_{11} &= u_1 + v_1 \\ c_{12} &= u_1 + v_2 \\ c_{22} &= u_2 + v_2 \\ c_{23} &= u_2 + v_3 \\ c_{33} &= u_3 + v_3 \\ c_{34} &= u_3 + v_4 \\ c_{44} &= u_4 + v_4 \\ c_{45} &= u_4 + v_5 \\ c_{55} &= u_5 + v_5 \end{aligned}\right\}$$

Λύνοντας το παραπάνω 10×10 γραμμικό σύστημα παίρνουμε εύκολα $U=\begin{bmatrix}3 & 5 & 6 & -65 & -80\end{bmatrix}$ και $V=\begin{bmatrix}0 & -6 & 37 & 65 & 73\end{bmatrix}$. Στη συνέχεια $\forall(i,j)\notin T$ θα έχουμε $s_{ij}=c_{ij}-u_i-v_j$, από όπου υπολογίζουμε τον πίνακα των μειωμένων κοστών S.Κάνοντας πράξεις λοιπόν παίρνουμε :

$$S=\begin{bmatrix} 0 & 0 & -21 & -58 & -103 \\ -5 & 0 & 0 & -24 & -1 \\ 64 & 32 & 0 & 0 & 11 \\ 99 & 72 & 28 & 0 & 0 \\ 81 & 93 & 77 & 80 & 0 \end{bmatrix}$$

Παρατηρείστε ότι για κελιά που αντιστοιχούν σε βασικά τόξα (i,j) του δέντρου είναι πάντα $s_{ij}=0$.Το αντίστροφο όμως δεν ισχύει, δηλαδή αν $s_{ij}=0$ τότε το τόξο (i,j) δεν είναι πάντα βασικό. Στην προκειμένη περίπτωση ένα τόξο (i,j) είναι βασικό εάν και μόνον αν $s_{ij}=0$ και $x_{ij}<\infty$.

2.2.2 Περιγραφή του Αλγορίθμου

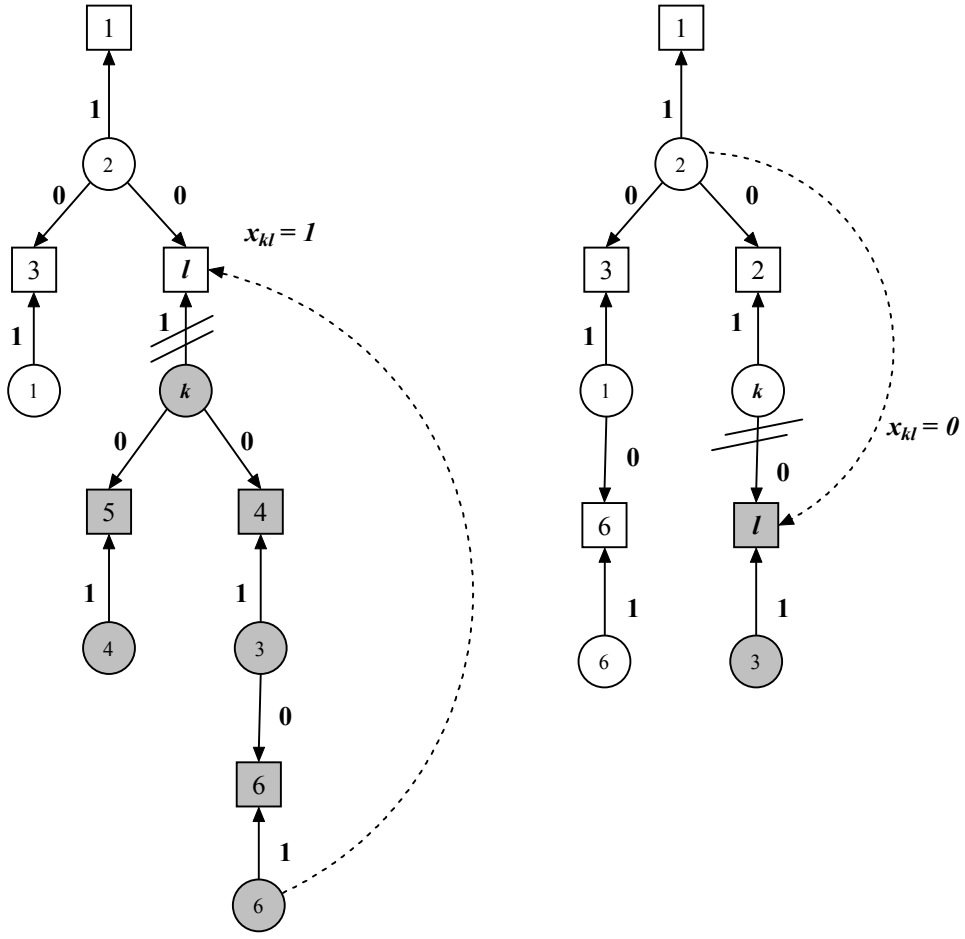
Ο πρωτεύων αλγόριθμος Simplex για το πρόβλημα Αντιστοίχησης δεν παρουσιάζει πολύ μεγάλες διαφορές από τον αντίστοιχο για το πρόβλημα Μεταφοράς. Η βασικότερη διαφορά του αλγορίθμου είναι η επιλογή του εισερχομένου τόξου, διαδικασία η οποία έχει κατά κάποιο τρόπο κωδικοποιηθεί και δεν απαιτείται στην ουσία ο προσδιορισμός των συνόλων C^+ και C^-.Η εξειδίκευση αυτή γίνεται προς επίτευξη καλύτερων υπολογιστικών επιδόσεων και είναι δυνατή λόγω της ειδικής μορφής του προβλήματος.

Έστω λοιπόν T το τρέχον δέντρο του αλγορίθμου. Αρχικά, ελέγχουμε τις τιμές s_{ij} των μειωμένων κοστών των μη βασικών τόξων του δέντρου. Αν είναι $s_{ij}\geq 0\ \ \forall(i,j)\notin T$ τότε ο αλγόριθμος τερματίζει και T είναι το βέλτιστο δέντρο, αλλιώς προσδιορίζει το εισερχόμενο τόξο (g,h) με βάση τη σχέση $s_{gh}=\delta=\min\{s_{ij}:s_{ij}<0\}$. Το εισερχόμενο τόξο ως γνωστό θα

δημιουργήσει έναν μοναδικό κύκλο C. Για την επιλογή του εξερχόμενου τόξου ακολουθείται η εξής αποτελεσματική τεχνική. Διακρίνουμε δύο περιπτώσεις:

- Αν ο κόμβος h βρίσκεται στο δρόμο $P[T,g]$ που ενώνει τον κόμβο g με τη ρίζα του δέντρου, τότε εξερχόμενο τόξο είναι το μοναδικό προς τα πάνω τόξο $(k,h)\in T$ που ανήκει στον κύκλο C.Σε αυτήν την περίπτωση είναι πάντα $x_{kh}(T)=1$.
- Αν ο κόμβος h δε βρίσκεται στο δρόμο $P[T,g]$ που ενώνει τον κόμβο g με τη ρίζα του δέντρου, τότε εξερχόμενο τόξο είναι το μοναδικό προς τα κάτω τόξο $(k,h)\in T$ που ανήκει στον κύκλο C.Σε αυτήν την περίπτωση είναι πάντα $x_{kh}(T)=0$.

Με άλλα λόγια το εξερχόμενο τόξο είναι πάντα αυτό που προσπίπτει στον κόμβο h και ανήκει στον κύκλου του δικτύου $T\cup(g,h)$.Εδώ πρέπει να παρατηρήσουμε ότι ο τρόπος επιλογής του εξερχόμενου τόξου βασίζεται και εξειδικεύεται εξ' ολοκλήρου ακολουθώντας τον κανόνα του Cunninghum.Οι δύο διαφορετικές περιπτώσεις επιλογής του εξερχόμενου τόξου φαίνονται στο παρακάτω σχήμα.



Εικόνα 2.15 – Οι δύο περιπτώσεις επιλογής του εξερχόμενου τόξου

Η ανανέωση του δέντρου γίνεται με τον ίδιο τρόπο ακριβώς που γίνεται και η ανανέωση του δέντρου για το πρόβλημα Μεταφοράς. Δηλαδή αν T το τρέχον δέντρο, (g,h) το εισερχόμενο τόξο και (k,ℓ) το εξερχόμενο τόξο, τότε το ανανεωμένο δέντρο T' θα είναι το δέντρο $T\cup(g,h)\sim(k,\ell)$. Οι τιμές $x_{ij}(T')$ ανανεώνονται κάθε φορά θέτοντας $x_{ij}(T')=1$, αν το τόξο (i,j) κατευθύνεται από τα φύλλα προς τη ρίζα, αλλιώς θέτουμε $x_{ij}(T')=0$.

Η ανανέωση των τιμών s_{ij} γίνεται πάλι με βάση τους κόμβους που ανήκουν στο αποκομμένο δέντρο $T*$ και ακολουθείται πάλι η ίδια διαδικασία, δηλαδή μεταβάλλονται

μόνο τα μειωμένα κόστη των τόξων που προσπίπτουν σε κόμβους που ανήκουν στο δέντρο T^*.

Στη συνέχεια θα παρουσιάσουμε τη βηματική περιγραφή του αλγορίθμου και θα εφαρμόσουμε τον αλγόριθμο για την συνέχεια της επίλυσης του παραδείγματος 2.3.

2.2.3 Βηματική Περιγραφή του Αλγορίθμου

Τα σημαντικότερα βήματα του αλγορίθμου είναι τα παρακάτω:

ΒΗΜΑ 0:*(Καθορισμός αρχικών μεταβλητών)*
Υπολόγισε το ισχυρό δέντρο ξεκινήματος T που είναι ένας δρόμος από τον κόμβο στήλη 1 μέχρι τον κόμβο γραμμή n καθώς και τις δυϊκές μεταβλητές των κόμβων του δέντρου θέτοντας $v_1 = 0$ και $s_{ij} = c_{ij} - u_i - v_j = 0$ για κάθε βασικό τόξο $(i, j) \in T$.

ΒΗΜΑ 1:*(Έλεγχος βελτιστότητας)*
Εάν $\forall (i, j) \notin T$ είναι $s_{ij} \geq 0$ τότε το δέντρο T είναι βέλτιστο και ο αλγόριθμος τερματίζει, αλλιώς πήγαινε στο βήμα 2.

ΒΗΜΑ 2:*(Επιλογή εισερχομένου τόξου)*
Υπολόγισε το εισερχόμενο τόξο (g,h) ελέγχοντας τα μειωμένα κόστη των μη βασικών τόξων με βάση τον τύπο $s_{gh} = \delta = \min\{s_{ij} : s_{ij} < 0\}$.

ΒΗΜΑ 3:*(Επιλογή εξερχόμενου τόξου)*
Έστω $P[T, g]$ ο δρόμος από τον κόμβο g προς στη ρίζα του δέντρου και C ο κύκλος που σχηματίζει το εισερχόμενο τόξο (g,h). Αν $h \in P[T, g]$ τότε εξερχόμενο είναι το τόξο $(k,h) \in C$ με $x_{kh} = 1$.Αντιθέτως αν $h \notin P[T, g]$, τότε εξερχόμενο είναι το τόξο $(k,h) \in C$ με $x_{kh} = 0$.

ΒΗΜΑ 4:*(Ανανέωση των μεταβλητών)*
Θέσε $x_{ij}(T') = 1$ αν το τόξο (i, j) κατευθύνεται από τα φύλλα του δέντρου προς τη ρίζα και $x_{ij}(T') = 0$ αλλιώς. Υπολόγισε το δέντρο T^* που είναι το δέντρο που αποκόβεται από τη ρίζα όταν αφαιρείται το εξερχόμενο τόξο. Αν $h \in T^*$ θέσε $q = -\delta > 0$, αλλιώς θέσε $q = \delta < 0$. Στη συνέχεια εάν ο κόμβος $b \in T^*$ είναι κόμβος-γραμμή θέσε $s_{\mathrm{b}j}(T') = s_{\mathrm{b}j}(T) - q$, $j = 1..n$ αλλιώς (δηλαδή ο κόμβος $b \in T^*$ είναι κόμβος-στήλη) θέσε $s_{i\,\mathrm{b}}(T') = s_{i\,\mathrm{b}}(T) + q$, $i = 1..n$.(Ανανεώνονται οι τιμές s_{ij} των τόξων (i, j) που προσπίπτουν σε κόμβους που ανήκουν στο δέντρο T^*)

ΒΗΜΑ 5:*(Ανανέωση του τρέχοντος δέντρου)*
Θέσε $T = T'$ και πήγαινε στο βήμα 1.

Στη συνέχεια θα παρουσιάσουμε την ακριβή επίλυση ενός παραδείγματος. Θα χρησιμοποιήσουμε τα δεδομένα που χρησιμοποιήσαμε στο προηγούμενο παράδειγμα(παράδειγμα 2.3).

Παράδειγμα 2.4

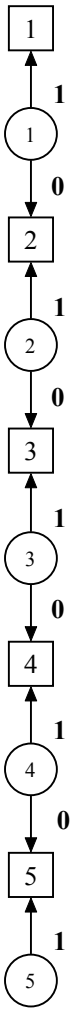
Δίνεται ο πίνακας κόστους

$$C = \begin{bmatrix} 3 & -3 & 19 & 10 & -27 \\ 0 & -1 & 42 & 46 & 77 \\ 70 & 32 & 43 & 71 & 90 \\ 34 & 1 & 0 & 0 & 8 \\ 1 & 7 & 34 & 65 & -7 \end{bmatrix}$$

για ένα 5×5 ***πρόβλημα Αντιστοίχησης. Να λυθεί το πρόβλημα χρησιμοποιώντας τον πρωτεύοντα αλγόριθμο Simplex.***

Λύση

Όλα τα αρχικά δεδομένα φαίνονται παρακάτω:



$$S = \begin{bmatrix} 0 & 0 & -21 & -58 & -103 \\ -5 & 0 & 0 & -24 & -1 \\ 64 & 32 & 0 & 0 & 11 \\ 99 & 72 & 28 & 0 & 0 \\ 81 & 93 & 77 & 80 & 0 \end{bmatrix}$$

Εικόνα 2.16 – Το εφικτό δέντρο του προβλήματος και η αντίστοιχη μήτρα μειωμένων κοστών.

Πίνακας 2.1.9 – Ο πίνακας που αντιστοιχεί στο δέντρο του σχήματος 2.16

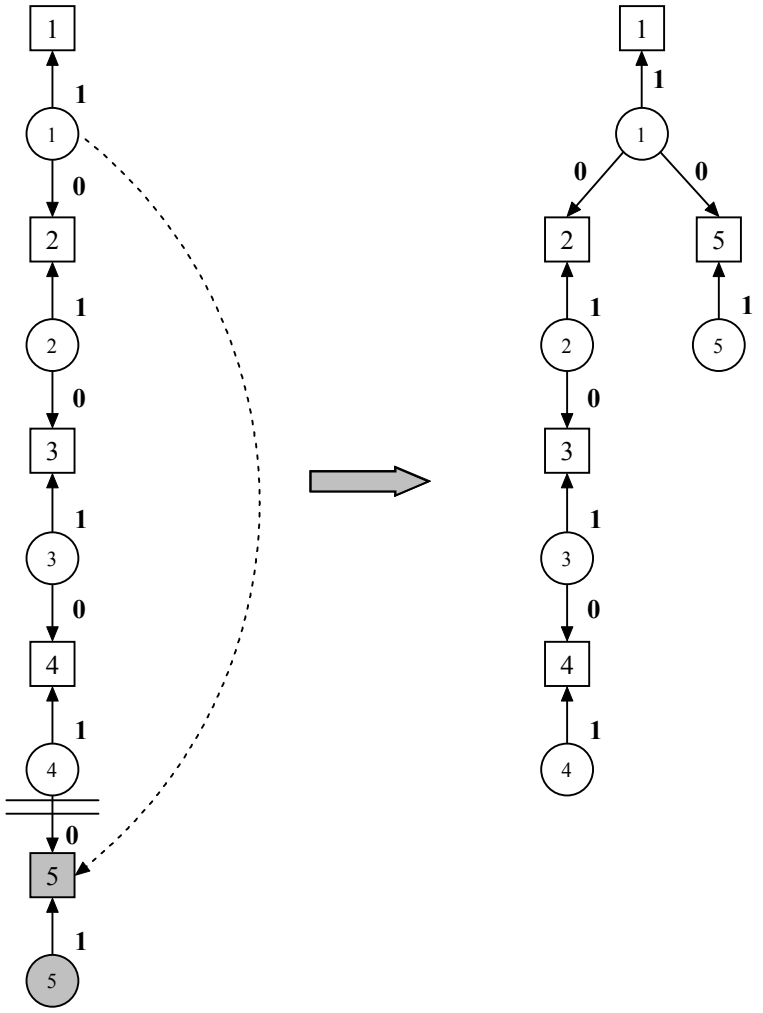
1	**0**	-21	-58	-103
-5	**1**	**0**	-24	-1
64	32	**1**	**0**	11
99	72	28	**1**	**0**
81	93	77	80	**1**

Παραπάνω παρουσιάζουμε τον πίνακα όπου φαίνονται όλες οι πληροφορίες συγκεντρωμένες, δηλαδή τον πίνακα που περιέχει και τις μεταβλητές απόφασης καθώς και

τα μειωμένα κόστη που αντιστοιχούν σε κάθε τόξο. Οι τιμές που βρίσκονται μέσα σε γκρίζα κελιά αντιπροσωπεύουν τις μεταβλητές απόφασης του προβλήματος. Θα μπορούσαν να παραληφθούν επειδή μπορούν εύκολα να υπολογισθούν από τη φορά των τόξων του δέντρου. Συμπεριλαμβάνονται όμως προς χάριν αισθητικής. Στη συνέχεια θα παρουσιασθούν λεπτομερώς οι επαναλήψεις του αλγορίθμου.

Επανάληψη 1

- Είναι $s_{gh} = \delta = \min\{s_{ij} : s_{ij} < 0\} = s_{15} = -103$. Συνεπώς $\delta = -103$ και $(g,h) = (1,5)$.
- Επειδή $h \notin P[T,g]$, κόβεται το μοναδικό προς τα κάτω τόξο (k,h) με $x_{kh} = 0$. Συνεπώς $(k,h) = (4,5)$.
- Συμβολίζουμε με T_r το σύνολο των κόμβων – γραμμών $b \in T*$ και με T_c το σύνολο των κόμβων στηλών $c \in T*$. Συνεπώς θα είναι $T_r = \{5\}$ και $T_c = \{5\}$.



1	**0**	-21	-58	-103	
-5	**1**	**0**	-24	-1	
64	32	**1**	**0**	11	
99	72	28	**1**	**0**	
81	93	77	80	**1**	**-103**
				+103	

→

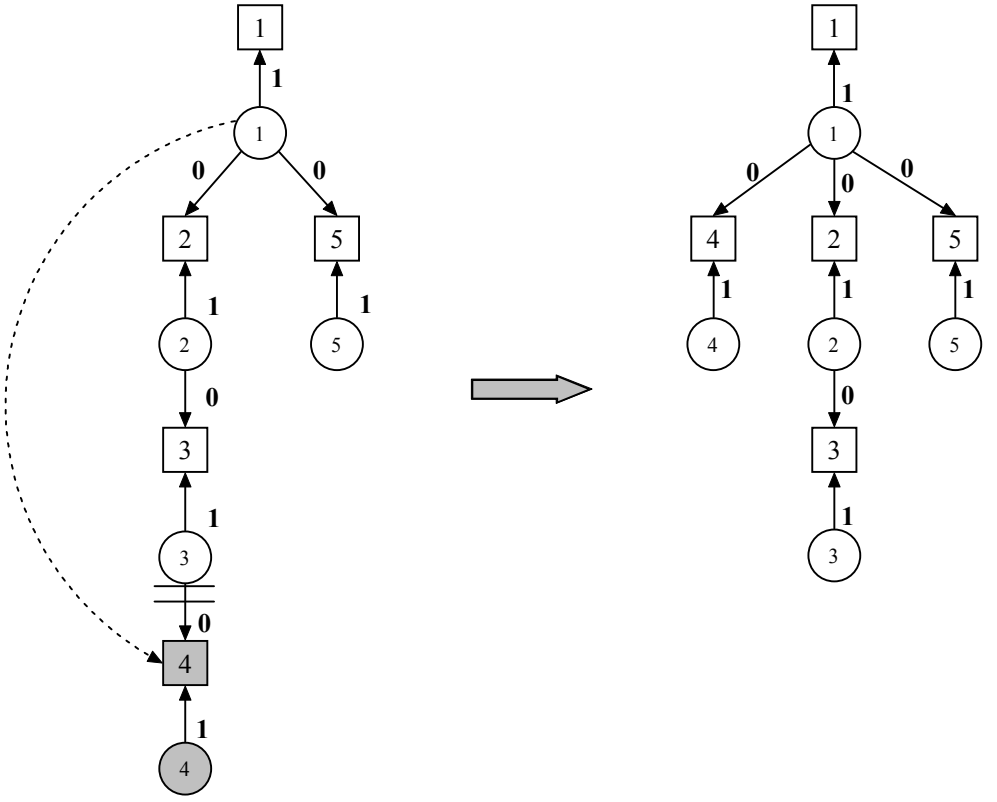
1	**0**	-21	-58	**0**
-5	**1**	**0**	-24	102
64	32	**1**	**0**	114
99	72	28	**1**	103
-22	-10	-26	-23	**1**

Εικόνα 2.17 – Η πρώτη επανάληψη του αλγορίθμου

Η ανανέωση των μεταβλητών s_{ij} με βάση τα σύνολα T_r, T_c και του τρέχοντος δέντρου φαίνεται αναλυτικά στο σχήμα 2.17. Οι μεταβλητές απόφασης x_{ij} ανανεώνονται με βάση τη φορά των αντίστοιχων τόξων. Παρατηρείστε ότι οι κόμβοι του υπόδεντρου $T*$ είναι χρωματισμένοι με γκρι χρώμα.

Επανάληψη 2

- Είναι $s_{gh} = \delta = \min\{s_{ij} : s_{ij} < 0\} = s_{14} = -58$. Συνεπώς $\delta = -58$ και $(g,h) = (1,4)$.
- Επειδή $h \notin P[T,g]$, κόβεται το μοναδικό προς τα κάτω τόξο (k,h) με $x_{kh} = 0$. Συνεπώς $(k,h) = (3,4)$.
- Επίσης είναι $T_r = \{4\}$ και $T_c = \{4\}$.
- Η ανανέωση των μεταβλητών s_{ij} με βάση τα σύνολα T_r, T_c και του τρέχοντος δέντρου φαίνεται αναλυτικά στα παρακάτω σχήματα. Οι μεταβλητές απόφασης x_{ij} ανανεώνονται με βάση τη φορά των αντίστοιχων τόξων.



1	**0**	-21	-58	**0**
-5	**1**	**0**	-24	102
64	32	**1**	**0**	114
99	72	28	**1**	103
-22	-10	-26	-23	**1**

-58

+58

1	**0**	-21	**0**	**0**
-5	**1**	**0**	34	102
64	32	**1**	58	114
41	14	-30	**1**	45
-22	-10	-26	35	**1**

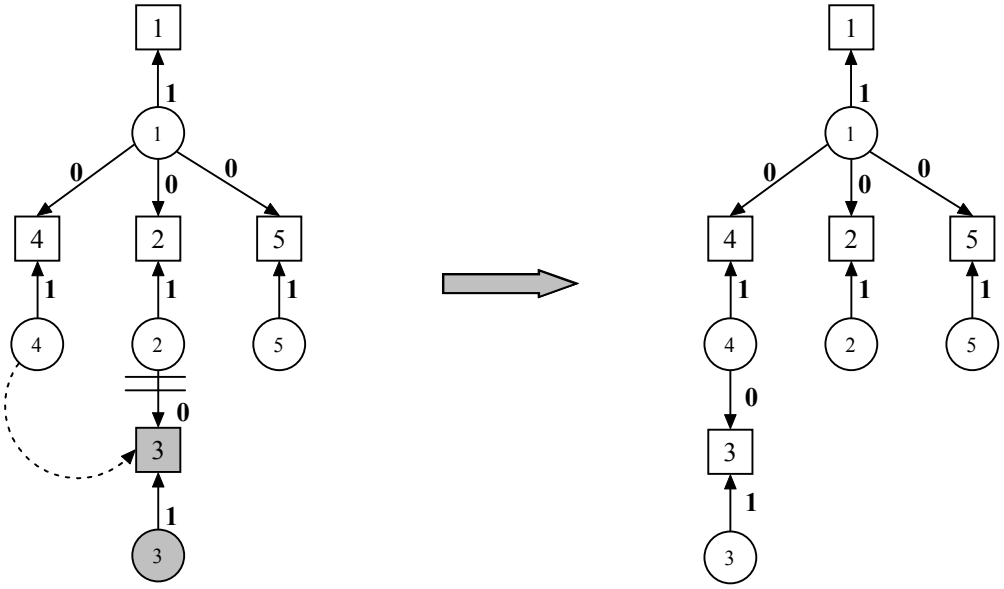
Εικόνα 2.18 – Η δεύτερη επανάληψη του αλγορίθμου

Παρατηρείστε ότι όταν η τιμή της μεταβλητής απόφασης του εξερχόμενου τόξου $x_{k\ell}$ είναι ίση με το μηδέν, οι μεταβλητές x_{ij} του ανανεωμένου δέντρου T' δεν μεταβάλλονται, δηλαδή όλα τα τόξα διατηρούν τη φορά που είχαν πριν, πράγμα πολύ φυσιολογικό αν σκεφτούμε το

γεγονός ότι ο αλγόριθμος Simplex για το πρόβλημα Αντιστοίχησης αποτελεί εξειδίκευση του αλγορίθμου Simplex για το πρόβλημα Μεταφοράς. Επίσης, η τιμή της αντικειμενικής συνάρτησης κατά τη διάρκεια τέτοιου είδους επαναλήψεων δεν μεταβάλλεται με αποτέλεσμα η επανάληψη να ονομάζεται εκφυλισμένη*(degenerate)*. Στην ουσία, τέτοιου είδους επαναλήψεις είναι «άσκοπες» αφού προσθέτουν υπολογιστικό έργο χωρίς να πετυχαίνουν βελτίωση της τιμής της αντικειμενικής συνάρτησης. Αυτή είναι μια πολύ σημαντική παρατήρηση αφού μια καλή τακτική ανάπτυξης αλγορίθμων Δικτυακού Προγραμματισμού αποτελεί η εύρεση μεθόδων αποφυγής εκφυλισμένων επαναλήψεων.

Επανάληψη 3

- Είναι $s_{gh} = \delta = \min\{s_{ij} : s_{ij} < 0\} = s_{43} = -30$. Συνεπώς $\delta = -30$ και $(g,h) = (4,3)$.
- Ο δρόμος $P[T,g]$ περιέχει τους κόμβους – στήλες 4,1.Επειδή όμως $h = 3$ συμπεραίνουμε ότι $h \notin P[T,g]$ και έτσι κόβεται το μοναδικό προς τα κάτω τόξο (k,h) με $x_{kh} = 0$. Συνεπώς $(k,h) = (2,3)$.
- Επίσης είναι $T_r = \{3\}$ και $T_c = \{3\}$.
- Η ανανέωση των μεταβλητών s_{ij} με βάση τα σύνολα T_r, T_c και του τρέχοντος δέντρου φαίνεται αναλυτικά στα παρακάτω σχήματα. Οι μεταβλητές απόφασης x_{ij} ανανεώνονται με βάση τη φορά των αντίστοιχων τόξων.



1	**0**	-21	**0**	**0**
-5	**1**	**0**	34	102
64	32	**1**	58	114
41	14	-30	**1**	45
-22	-10	-26	35	**1**

+30

-30

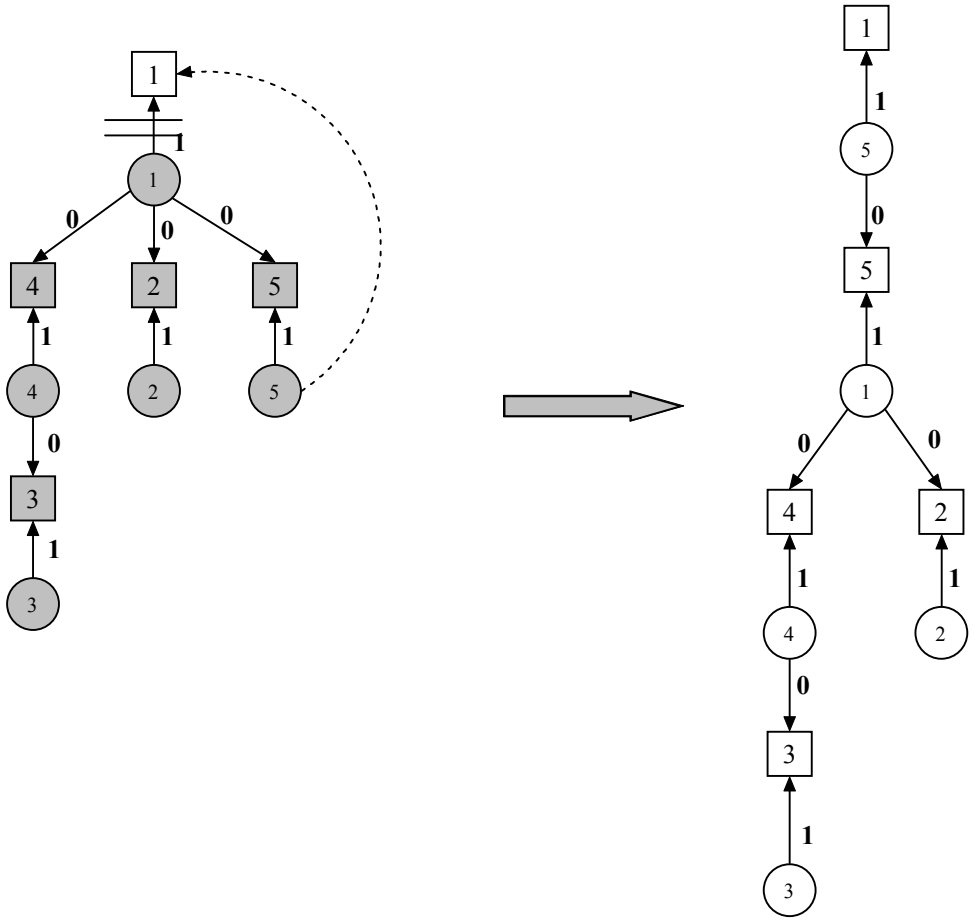
1	**0**	9	**0**	**0**
-5	**1**	30	34	102
34	2	**1**	28	84
41	14	**0**	**1**	45
-22	-10	4	35	**1**

Εικόνα 2.19 – Η τρίτη επανάληψη του αλγορίθμου

Μέχρι αυτή τη στιγμή, οι τρεις επαναλήψεις που εκτέλεσε ο αλγόριθμος ήταν του ίδιου τύπου και συγκεκριμένα αντιστοιχούν στο δεξιό σχήμα του σχήματος 2.15. Ως εκ τούτου, η τιμή της αντικειμενικής συνάρτησης παραμένει σταθερή και ίση με $z(T) = \sum_{(i,j)\in T} c_{ij} x_{ij} = 38$.

Επανάληψη 4

- Είναι $s_{gh} = \delta = \min\{s_{ij} : s_{ij} < 0\} = s_{51} = -22$. Συνεπώς $\delta = -22$ και $(g,h) = (5,1)$.
- Ο δρόμος $P[T,g]$ περιέχει τους κόμβους – στήλες 5,1.Επειδή όμως $h = 1$ συμπεραίνουμε ότι $h \in P[T,g]$ και έτσι κόβεται το μοναδικό προς τα πάνω τόξο (k,h) με $x_{kh} = 1$. Συνεπώς $(k,h) = (1,1)$.Πρόκειται για μη εκφυλισμένη επανάληψη.
- Επίσης είναι $T_r = \{1,2,3,4,5\}$ και $T_c = \{2,3,4,5\}$.Η ανανέωση των μεταβλητών s_{ij} και x_{ij} γίνεται με τις γνωστές αλγοριθμικές διαδικασίες που ξέρουμε όπως παρακάτω:



1	**0**	9	**0**	**0**	+22
-5	**1**	30	34	102	+22
34	2	**1**	28	84	+22
41	14	**0**	**1**	45	+22
-22	-10	4	35	**1**	+22
	-22	**-22**	**-22**	**-22**	

22	**0**	9	**0**	**1**
17	**1**	30	34	102
56	2	**1**	28	84
63	14	**0**	**1**	45
1	-10	4	35	**0**

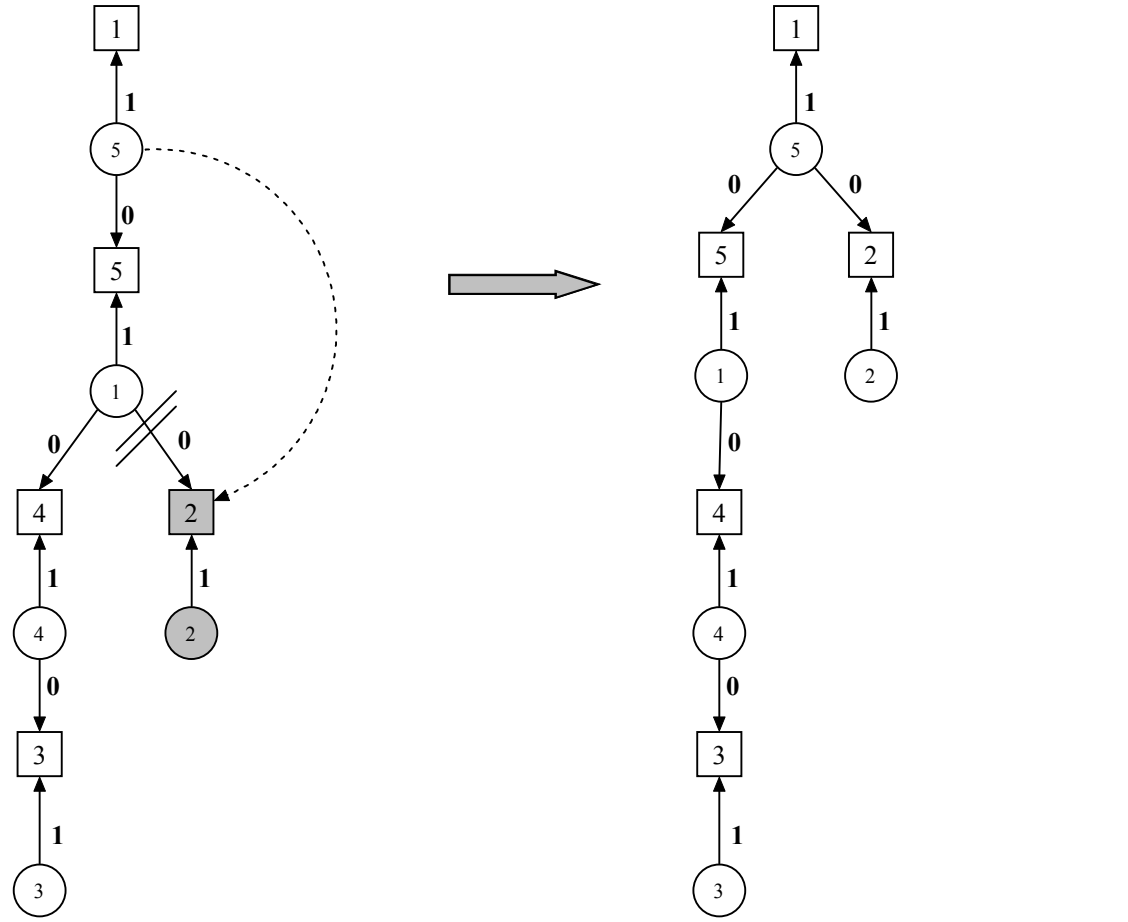
Εικόνα 2.20 – Η τέταρτη επανάληψη του αλγορίθμου

Επανάληψη 5

- Είναι $s_{gh} = \delta = \min\{s_{ij} : s_{ij} < 0\} = s_{52} = -10$. Συνεπώς $\delta = -10$ και $(g,h) = (5,2)$.
- Ο δρόμος $P[T,g]$ περιέχει μόνο τον κόμβο – στήλη 1.Επειδή όμως $h = 2$ συμπεραίνουμε ότι $h \notin P[T,g]$ και έτσι κόβεται το μοναδικό προς τόξο (k,h) που

έχει κατεύθυνση από τη ρίζα προς τα φύλλα του δέντρου, ανήκει στον σχηματιζόμενο κύκλο C και έχει $x_{kh} = 0$. Συνεπώς $(k,h) = (1,2)$.Πρόκειται για μια εκφυλισμένη επανάληψη.

- Επίσης είναι $T_r = \{2\}$ και $T_c = \{2\}$.Η αναλυτική ανανέωση των μεταβλητών T, s_{ij} και x_{ij} φαίνεται παρακάτω:



22	**0**	9	**0**	
17	**1**	30	34	
56	2	**1**	28	
63	14	**0**	**1**	
1	-10	4	35	

-10

+10

22	10	9	**0**	**1**
7	**1**	20	24	92
56	12	**1**	28	84
63	24	**0**	**1**	45
1	**0**	4	35	**0**

Εικόνα 2.20 – Η πέμπτη επανάληψη του αλγορίθμου και το τελικό βέλτιστο δέντρο

Αν παρατηρήσουμε προσεκτικά τον τελικό πίνακα που προέκυψε, παρατηρούμε ότι $s_{ij} \geq 0 \ \ \forall (i,j) \notin T$. Αυτό σημαίνει ότι ο αλγόριθμος τερματίζει και ότι το τελικό δέντρο που προέκυψε είναι βέλτιστο.

Η λύση που έχει παραχθεί μπορεί να παρασταθεί και με το βέλτιστο διάνυσμα ανάθεσης, το οποίο για το συγκεκριμένο πρόβλημα θα είναι το $p = [5 \quad 2 \quad 3 \quad 4 \quad 1]$. Άρα, αν το συγκεκριμένο πρόβλημα απευθυνόταν σε ένα πρόβλημα ανάθεσης έργων, όπου ψάχναμε τη βέλτιστη αντιστοίχηση ατόμων και εργασιών με δεδομένα τα διάφορα κόστη, το βέλτιστο κόστος θα προέκυπτε από την αντιστοίχιση της εργασίας p_i στο άτομο

i , $i = 1..n$, όπου $n = 5$. Επίσης, τελική και βέλτιστη τιμή της αντικειμενικής συνάρτησης θα είναι $z(T) = \sum_{(i,j)\in T} c_{ij} x_{ij} = 16$.

2.2.4 Προγραμματισμός του Αλγορίθμου

Στη συνέχεια θα παρουσιάσουμε τον έναν αποτελεσματικό τρόπο προγραμματισμού του αλγορίθμου. Οι δομές δεδομένων που θα χρησιμοποιηθούν είναι οι ίδιες που χρησιμοποιήθηκαν και στο πρόβλημα μεταφοράς. Μάλιστα, η διαχείρισή τους είναι ακόμη πιο εύκολη αφού θέτουμε $m = n$.

Αλγόριθμος A1

Ο παρακάτω αλγόριθμος λύνει το πρόβλημα Αντιστοίχησης, χρησιμοποιώντας τον πρωτεύοντα αλγόριθμο Simplex.Ξεκινάει με ένα εφικτό δέντρο ξεκινήματος που είναι ένας δρόμος από τον κόμβο-στήλη 1 προς τον κόμβο γραμμή n.

Είσοδος: *Ο πίνακας κόστους* **C(nxn)**.

Έξοδος : *Η λύση του προβλήματος Αντιστοίχησης* **X(nxn)**.

```
1.  [X,S]=FeasibleTree(n);
2.  [U,V]=DualVariables(S,C,n);
3.  S=ReducedCosts(n,C,U,V);
4.  p=InitializeNodeFather(X,n);
5.  d=InitializeNodeDepth(p,n);
6.  t=InitializeNodeDirection(n);
7.  min=-inf;
8.  simplex=1;
9.  iterationsA1=0;
10. while simplex==1
11.   [min,g,h]=EnteringArc(S,n);
12.   if min~=0
13.     simplex=1;
14.     [k,l,Tcut,upwards]=LeavingArc(g,h,p,n);
15.     [e1,e2,f1,z]=Steam(p,g,h,k,l,d,n,Tcut);
16.     d=UpdateNodeDepth(p,n,Tcut,z,d,e1,e2,f1);
17.     p=UpdateNodeFather(p,z,e2,f1);
18.     [S,X]=UpdateVariables(t,p,d,n,X,g,h,k,l,Tcut,min,S,upwards);
19.   else
20.     simplex=0;
21.   end
22. end
```

Αλγόριθμος 2.2.1 – Ο κώδικας του πρωτεύοντος αλγορίθμου Simplex για το πρόβλημα Αντιστοίχησης

Είναι εμφανές ότι ο παραπάνω αλγόριθμος παρουσιάζει πολλές ομοιότητες με τον αντίστοιχο αλγόριθμο Simplex για το πρόβλημα Μεταφοράς. Αν παρατηρήσουμε τον κώδικα προσεχτικά, θα δούμε ότι πολλές συναρτήσεις είναι ίδιες και το μόνο που αλλάζει είναι οι παράμετροι που περνιούνται σε αυτές. Έτσι , αρχικά υπολογίζεται το εφικτό δέντρο-δρόμος με τη διαδικασία **`FeasibleTree()`**. Στη συνέχεια υπολογίζονται οι δυϊκές μεταβλητές u_i, v_j $i, j = 1..n$, τα μειωμένα κόστη s_{ij} και αρχικοποιούνται τα διανύσματα πατέρα – κόμβου p, βάθους d και κατεύθυνσης t. Μετά υπολογίζονται το εισερχόμενο και εξερχόμενο τόξο και ανανεώνονται οι δομές που χρησιμοποιεί ο αλγόριθμος. Ο αλγόριθμος θα τερματίσει όταν η διαδικασία **`EnteringArc()`** δεν εντοπίσει $s_{ij} < 0$ και συνεπώς θα επιστρέψει τιμή 0 στη μεταβλητή **`min`**. Οι διαδικασίες που κρίναμε σκόπιμο να αναλύσουμε είναι οι παρακάτω:

- **[X, S] = FeasibleTree(n)**

Με αυτή τη διαδικασία κατασκευάζουμε το εφικτό δέντρο για το πρόβλημα Αντιστοίχησης όπως αυτό περιγράφηκε στην εισαγωγή που κάναμε για τον αλγόριθμο. Στην ουσία,

δεδομένου του μεγέθους του προβλήματος n, σχηματίζουμε δύο $n \times n$ μήτρες X, S όπου αποθηκεύουμε τις αρχικές πληροφορίες για τις μεταβλητές απόφασης και τα μειωμένα κόστη που αντιστοιχούν στο αρχικό δέντρο T. Συγκεκριμένα, θέτουμε, $X(i,i)=1$, $i=1..n$ (τόξα που κατευθύνονται από τα φύλλα προς τη ρίζα του δέντρου), $X(i,i+1)=0$, $i=1..n-1$ (τόξα που κατευθύνονται από τη ρίζα προς τα φύλλα του δέντρου), αλλιώς θέτουμε $X(i,j)=\infty$ (μη βασικό τόξο).Επίσης $\forall (i,j) \in T$ θέτουμε $S(i,j)=0$, αλλιώς θέτουμε $S(i,j)=\infty$. Γενικά, το σύμβολο άπειρο(∞) θα το χρησιμοποιούμε για να δηλώσουμε την ανυπαρξία του συγκεκριμένου τόξου. Ο συγκεκριμένος αλγόριθμος φαίνεται παρακάτω:

Αλγόριθμος FeasibleTree
Ο αλγόριθμος που υπολογίζει το εφικτό δέντρο ξεκινήματος.
Είσοδος: *Η διάσταση του προβλήματος* **n.**
Έξοδος : *Οι αρχικοποιημένες μήτρες* **X(n, n),S(n, n)**

```
1.  function [X,S]=FeasibleTree(n)
2.  for i=1:n
3.    for j=1:n
4.      if i==j
5.        X(i,j)=1;
6.        S(i,j)=0;
7.      elseif i+1==j
8.        X(i,j)=0;
9.        S(i,j)=0;
10.     else
11.       X(i,j)=inf;
12.       S(i,j)=inf;
13.     end
14.   end
15. end
```

Αλγόριθμος 2.2.2 – Ο αλγόριθμος κατασκευής του εφικτού δέντρου ξεκινήματος

Σε αυτό το σημείο πρέπει να κάνουμε μια σημαντική παρατήρηση. Ο παραπάνω αλγόριθμος θα μπορούσε να εκφραστεί με έναν αρκετά πιο αποδοτικό τρόπο, λαμβάνοντας υπόψη ότι οι έλεγχοι γενικά καταναλώνουν μεγάλο μέρος του υπολογιστικού χρόνου ενός αλγορίθμου. Συνεπώς, μια πιο «κομψή» έκδοση του παραπάνω αλγορίθμου θα ήταν η εξής:

Αλγόριθμος FeasibleTreeOp
Ο αλγόριθμος που υπολογίζει το εφικτό δέντρο ξεκινήματος με αποτελεσματικότερο τρόπο.
Είσοδος: *Η διάσταση του προβλήματος* **n.**
Έξοδος : *Οι αρχικοποιημένες μήτρες* **X(n, n),S(n, n)**

```
1.  X(1:n,1:n)=inf;
2.  S(1:n,1:n)=inf;
3.  X(n,n)=1;
4.  S(n,n)=0;
5.  for i=1:n-1
6.    X(i,i)=1;
7.    S(i,i)=0;
8.    X(i,i+1)=0;
9.    S(i,i+1)=0;
10. end
```

Αλγόριθμος 2.2.3 – Ένας αποδοτικότερος αλγόριθμος κατασκευής του εφικτού δέντρου ξεκινήματος

Παρατηρείστε ότι και θεωρητικά ο αλγόριθμος 2.2.3 είναι καλύτερος από τον αλγόριθμο 2.2.2. Και αυτό συμβαίνει γιατί, αν θεωρήσουμε ότι οι εντολές των γραμμών 1,2 του αλγορίθμου 2.2.3 είναι της τάξης $O(1)$, ο αλγόριθμος 2.2.3 τρέχει σε χρόνο $O(n)$, ενώ ο

αλγόριθμος 2.2.2 τρέχει προσεγγιστικά σε χρόνο $O(n^2)$.Η παρατήρηση αυτή έγινε για να δούμε πως μπορούμε να κάνουμε βελτιώσεις σε έναν κώδικα. Στην πραγματικότητα, επειδή η διαδικασία **`FeasibleTree()`** καλείται μόνο μια φορά από το κυρίως πρόγραμμα του αλγορίθμου, δεν επηρεάζει το χρόνο εκτέλεσης όλου του αλγορίθμου.

- **p = InitializeNodeFather(n)**

Με αυτή τη διαδικασία αρχικοποιούμε το διάνυσμα πατέρα – κόμβου, δηλαδή το διάνυσμα εκείνο που αντιστοιχεί στο εφικτό δέντρο ξεκινήματος. Το διάνυσμα αυτό όπως έχουμε αναφέρει εκφράζει τη «δομή» του δέντρου και είναι πολύ σημαντικό για την εκτέλεση του αλγορίθμου. Παρακάτω φαίνεται ο κώδικας που γράψαμε:

Αλγόριθμος InitializeNodeFather
Ο αλγόριθμος που υπολογίζει το εφικτό δέντρο ξεκινήματος.
Είσοδος: *Η διάσταση του προβλήματος* **n.**
Έξοδος : *Το αρχικοποιημένο διάνυσμα* **p(2n)**.

```
1. function p=InitializeNodeFather(n)
2. p=0;
3. p(n+1)=-1;
4. for i=1:n
5. p(i)=n+i;
6. end
7. for i=n+2:2*n
8. p(i)=i-n-1;
9. end
```

Αλγόριθμος 2.2.4 – Ο αλγόριθμος αρχικοποίησης του διανύσματος πατέρα-κόμβου.

- **[k, l, Tcut, upwards] = LeavingArc (g , h , p , n)**

Αλγόριθμος LeavingArc
Ο αλγόριθμος που υπολογίζει το εξερχόμενο τόξο.
Είσοδος: *Το εισερχόμενο τόξο* **(g,h),** *το διάνυσμα πατέρα-κόμβου* **p(2n)** *και η διάσταση του προβλήματος* **n.**
Έξοδος : *Το εξερχόμενο τόξο* **(k,l),** *το δέντρο* **T***,*και η μεταβλητή φοράς του εξερχόμενου τόξου* **upwards**.

```
1. function [k,l,Tcut,upwards]=LeavingArc(g,h,p,n)
2. l=h;
3. s=g;
4. i=1;
5. upwards=0;
6. while s~=n+1
7.   u=s;
8.   i=i+1;
9.   s=p(s);
10.  if s==h+n
11.    s=n+1;
12.    upwards=1;
13.  end
14.end
15.if upwards==1
16.  k=u;
17.  start=k;
18.else
19.  k=p(h+n);
20.  start=l+n;
21.end
22.Tcut=Preorder(start,p,n);
```

Αλγόριθμος 2.2.5 – Υπολογισμός εξερχόμενου τόξου.

Με αυτή τη διαδικασία υπολογίζουμε κάθε φορά το εξερχόμενο τόξο (k,ℓ). Ως μεταβλητές εισόδου αυτής της διαδικασίας περνιούνται το εισερχόμενο τόξο (g,h), το διάνυσμα πατέρα κόμβου p και η διάσταση του προβλήματος n.

Η διαδικασία υπολογίζει επίσης τον δέντρο T^* καλώντας τη διαδικασία **Preorder()**,όπως αυτήν παρουσιάσθηκε και στο πρόβλημα Μεταφοράς καθώς και μία μεταβλητή **upwards** που παίρνει τιμές στο διάστημα $\{0,1\}$ και υποδηλώνει τη φορά του εξερχόμενου τόξου. Δηλαδή, αν $upwards=1$ τότε το εξερχόμενο τόξο κατευθύνεται από τα φύλλα του δέντρου προς τη ρίζα, ενώ αν $upwards=0$, τότε το εξερχόμενο τόξο (k,ℓ) κατευθύνεται από τη ρίζα προς τα φύλλα του δέντρου.Αν παρατηρήσουμε προσεχτικά τον κώδικά του αλγορίθμου, θα δούμε ότι υλοποιείται ακριβώς η διαδικασία που περιγράφηκε για την επιλογή του εξερχόμενου τόξου. Αρχικά, ελέγχεται αν ο κόμβος h ανήκει στο δρόμο που διαγράφεται από τον κόμβο g προς τη ρίζα του δέντρου και αναλόγως καθορίζουμε την τιμή της μεταβλητής **upwards**(γραμμές 5-14). Στη συνέχεια, ακολουθώντας όσα αναφέρθηκαν σχετικά με την επιλογή του εξερχόμενου τόξου, η μεταβλητή **upwards** μας οδηγεί στον υπολογισμό των μεταβλητών k και ℓ (γραμμές 15-21).

- **[S, X]=UpdateVariables(t ,p ,d ,n ,X ,g ,h ,k ,l ,Tcut ,min ,S ,upwards)**

Αλγόριθμος UpdateVariables
Ο αλγόριθμος ανανέωσης των μειωμένων κοστών και μεταβλητών απόφασης.
Είσοδος: t(2n), p(2n), d(2n), n,X(n,n), g, h, k, l, Tcut, min, S,upwards
Έξοδος : *Το εξερχόμενο τόξο* **(k,l),** *το δέντρο* **T***,*και η μεταβλητή φοράς του εξερχόμενου τόξου* **upwards**.

```
1.  function [S,X]=UpdateVariables(t,p,d,n,X,g,h,k,l,Tcut,min,S,upwards)
2.  for i=1:n
3.    if (mod(d(i),2)==1)&(X(i,p(i)-n)~=inf)
4.      X(i,p(i)-n)=1;
5.    elseif (mod(d(i),2)==0)&(X(i,p(i)-n)~=inf)
6.      X(i,p(i)-n)=0;
7.    end
8.  end
9.  for i=n+2:2*n
10.   if (mod(d(i),2)==1)&(X(p(i),i-n)~=inf)
11.     X(p(i),i-n)=1;
12.   elseif (mod(d(i),2)==0)&(X(p(i),i-n)~=inf)
13.     X(p(i),i-n)=0;
14.   end
15. end
16. X(g,h)=upwards;
17. X(k,l)=inf;
18. for i=1:length(Tcut)
19.   if h+n==Tcut(i)
20.     min=-min;
21.   end
22. end
23. for i=1:length(Tcut)
24.   if Tcut(i)<=n
25.     for j=1:n
26.       S(Tcut(i),j)=S(Tcut(i),j)-min;
27.     end
28.   else
29.     for j=1:n
30.       S(j,Tcut(i)-n)=S(j,Tcut(i)-n)+min;
31.     end
32.   end
33. end
```

Αλγόριθμος 2.2.6 – Ο αλγόριθμος ανανέωσης των πινάκων X,S

Παρόλο που περιγράψαμε την αντίστοιχη διαδικασία ανανέωσης των πινάκων S και X για το πρόβλημα Μεταφοράς, κρίναμε σκόπιμο να παρουσιάσουμε τον αλγόριθμο ανανέωσης και για το πρόβλημα Αντιστοίχησης λόγω ενός διαφορετικού τρόπου ανανέωσης του πίνακα X.Παρόλο που ο πίνακας X θα μπορούσε να μην ανανεώνεται, αλλά να υπολογιστεί μια φορά στο τέλος ανάλογα με τη φορά των τόξων του βέλτιστου δέντρου, θεωρήσαμε σωστό να ανανεώνουμε αυτές τις τιμές για να έχουμε μια γενική εικόνα της κατάστασης των μεταβλητών του αλγορίθμου. Όσον αφορά την ανανέωση των μεταβλητών απόφασης, αυτή γίνεται με βάση τα βάθη των κόμβων του δέντρου, δηλαδή ανανεώνουμε τις μεταβλητές απόφασης, δεδομένου ότι οι κόμβοι – γραμμές βρίσκονται πάντα σε περιττά βάθη($d = 1,3,...,2k+1$) και οι κόμβοι στήλες σε άρτια βάθη ($d = 0,2,..,2k$).Αυτό φαίνεται αναλυτικά στις γραμμές 2-15 του κώδικα.

2.3 ΑΝΑΦΟΡΕΣ

[Dn1] Dantzig, G.B.,"*Linear Programming and Extensions*", Princeton NJ, Princeton University Press

[Dn2] Dantzig, G.B., 1951,*"Application of the simplex method to a transportation problem",* In Activity Analysis and Production and Allocation, edited by T.C. Koopmans, Willey, N.Y. (359-373)

[Cun] Cunningham, W.H. (1976),"*A Network Simplex Method"*, Math Prog 11,105-116

[BGK] R.Barr, F.Glover, D.Klingman,"*An alternating path basis for assignment problems",* Math. Programming 13(1977),1-13

[Mn] Minieka, (1978),"*Optimization Algorithms for Networks and Graphs*" New York: Marcel Dekker

[JB] Jensen and Barnes, (1980),*"Network Flow Programming"* Malabar, Fla.: R.E. Krieger Pub. Co. , 1980

[B1] Bertsekas D.**, "***Network Optimization: Continuous and Discrete Models"* Athena Scientific, 1998

[CP] Papadimitriou C., *"Combinatorial Optimization – Algorithms and Complexity"*,Prentice Hall (1982)

[Pap1] K. Paparrizos, *"Linear Programming – Algorithms and Applications"*, Zygos Publications, Thessaloniki 1999, in Greek

[Kn] D.E. Knuth, *"The Art of Computer Programming – Fundamental Algorithms"*, 3rd edition, Addison-Wesley, 1997

[Pap2] K. Paparrizos, *"Network Programming"*, Lecture notes in Greek

[Pap3] K. Paparrizos, *"Algorithms – Analysis, Design and Complexity"*, Lecture notes in Greek

[SS] Stephanides G., Samaras N., *"Computational Methods with Matlab"*, Zygos Publications, Thessaloniki 1999, in Greek

[PK] Paparrizos K., *"Matlab 6"*, Zygos Publications, Thessaloniki 2001, in Greek

ΚΕΦΑΛΑΙΟ 3

Ο ΔΕΝΔΡΙΚΟΣ ΑΛΓΟΡΙΘΜΟΣ ΠΑΠΑΡΡΙΖΟΥ

(ΞΕΚΙΝΗΜΑ ΜΕ ΔΕΝΤΡΟ BALINSKI)

Στο κεφάλαιο αυτό θα παρουσιάσουμε έναν νέο αλγόριθμο για την επίλυση των προβλημάτων Μεταφοράς και Αντιστοίχησης. Πρόκειται για τον δενδρικό αλγόριθμο Παπαρρίζου που χρησιμοποιεί ως εφικτό δέντρο ξεκινήματος το δέντρο Balinski. Το δέντρο αυτό κατασκευάστηκε πρώτη φορά από τον M.L. Balinski**[B3]** και γι' αυτό φέρει το όνομά του. Ο αλγόριθμος που θα περιγραφεί είναι ένας αλγόριθμος εξωτερικών σημείων, δηλαδή χρησιμοποιεί εξωτερικά σημεία της εφικτής περιοχής για να φτάσει στο βέλτιστο σημείο. Σε αλγορίθμους τέτοιου είδους, σημαντικό ρόλο παίζει κάθε φορά η αρχική λύση που θα χρησιμοποιήσουμε. Επηρεάζει άμεσα τον αριθμό των επαναλήψεων του αλγορίθμου και συνεπώς και το χρόνο εκτέλεσης του αλγορίθμου. Όπως θα δούμε και στο κεφάλαιο της υπολογιστικής μελέτης, το δέντρο Balinski δεν αποτελεί αποδοτική από υπολογιστική άποψη αρχική λύση. Συνεπώς, εξαναγκάζει τον αλγόριθμο σε περισσότερες επαναλήψεις ,αυξάνοντας κατ' αυτόν τον τρόπο το χρόνο εκτέλεσης του αλγορίθμου.

3.1 ΕΦΑΡΜΟΓΗ ΣΤΟ ΠΡΟΒΛΗΜΑ ΜΕΤΑΦΟΡΑΣ

3.1.1 Περιγραφή του Δέντρου Balinski για το πρόβλημα Μεταφοράς

Ο αλγόριθμος που θα περιγράψουμε επιλύει ένα $m \times n$ πρόβλημα Μεταφοράς και ξεκινάει με το δέντρο Balinski, το οποίο είναι δυϊκά εφικτό και έχει ειδική δομή. Το δέντρο Balinski έχει πάντα ρίζα τον κόμβο γραμμή 1(επίπεδο 0).Όλοι οι κόμβοι στήλες βρίσκονται στο επίπεδο 1 ενώ οι κόμβοι γραμμές εκτός της ρίζας βρίσκονται στο επίπεδο 2.Δηλαδή το δέντρο περιέχει τόξα της μορφής $(1, j)$, $j = 1,...,n$ και (i, j) , $i = 2,...,m$, j είναι κόμβος στήλη. Επειδή μερικά από τα βασικά τόξα του δέντρου T είναι πάντα της μορφής $(1, j)$, $j = 1,...,n$ και ο κόμβος γραμμή 1 είναι η ρίζα του δέντρου μπορούμε εύκολα να υπολογίσουμε τις δυϊκές μεταβλητές της ρίζας του δέντρου και όλων των κόμβων στηλών θέτοντας $u_1(T) = 0$ και $s_{1j}(T) = 0, j = 1,...,n$. Αναπτύσσοντας την τελευταία σχέση έχουμε

$$s_{1j}(T) = 0 \Rightarrow c_{1j} - u_1(T) - v_j(T) = 0 \Rightarrow v_j(T) = c_{1j}$$

Τη στιγμή αυτή έχουν υπολογιστεί οι δυϊκές μεταβλητές των κόμβων στηλών. Για να εντοπίσουμε τα υπόλοιπα τόξα του δέντρου θέτουμε:

$$u_i(T) = \min\left\{c_{ij} - v_j(T) : j = 1,...,n\right\} \quad ,i = 2,...,m$$

Έστω ότι μετά την εφαρμογή του παραπάνω τύπου προκύπτει για κάθε γραμμή $i = 2,...,m$ μια στήλη $j(i)$ ώστε:

$$c_{ij(i)} - v_{j(i)}(T) = \min\left\{c_{ij} - v_j(T) : j = 1,...,n\right\} \quad ,i = 2,...,m$$

Τότε το τόξο $(i, j(i))$ είναι βασικό τόξο του δέντρου Balinski και προφανώς θα ισχύει η παρακάτω σχέση:

$$u_i(T) = c_{ij(i)} - v_{j(i)}(T) \ ,i = 2,...,m$$

Στο σημείο αυτό έχουν υπολογιστεί όλες οι δυϊκές μεταβλητές των κόμβων του δέντρου και συνεπώς εύκολα από εδώ και πέρα υπολογίζονται τα μειωμένα κόστη $s_{ij}(T)$ με βάση τον γνωστό τύπο:

$$s_{ij}(T) = c_{ij} - u_i(T) - v_j(T)$$

Στη συνέχεια μένει να υπολογίσουμε τις μεταβλητές απόφασης x_{ij} των βασικών τόξων του δέντρου. Για να γίνουν αυτοί οι υπολογισμοί θα πρέπει να χρησιμοποιήσουμε το m-διάστατο διάνυσμα προσφοράς a και το n-διάστατο διάνυσμα ζήτησης b που μας δίνονται ως δεδομένα του προβλήματος. Επειδή το δέντρο έχει τρία επίπεδα, 0,1 και 2 και όλοι οι κόμβοι γραμμές του δέντρου εκτός της ρίζας είναι φύλλα, θέτουμε:

$$x_{ij}(T) = a(i)\ ,\ i \neq 1\ ,\ (i,j) \in T$$

Για τα υπόλοιπα τόξα $(1, j)$ θα έχουμε $x_{1j}(T) = b(j)$, αν j είναι φύλλο του δέντρου, ενώ αν j δεν είναι φύλλο θα έχουμε:

$$x_{1j}(T) = b(j) - \sum_{(i,j)\in T,\ i\neq 1} x_{ij}$$

Με αυτό τον τρόπο ικανοποιούνται οι περιορισμοί του προβλήματος Μεταφοράς όπως αυτοί περιγράφηκαν στο εισαγωγικό κεφάλαιο. Σε αυτό το σημείο προσδιορίσθηκε πλήρως το εφικτό δέντρο ξεκινήματος Balinski.

Παράδειγμα 3.1

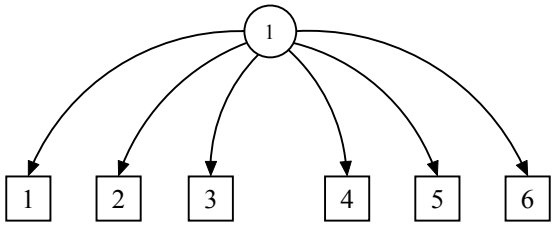
Δίνεται ο πίνακας κόστους

$$C = \begin{bmatrix} -5 & 49 & 28 & 20 & 47 & 43 \\ 19 & 62 & 32 & 61 & 0 & 13 \\ -2 & 63 & 14 & 46 & 28 & 53 \\ 1 & 50 & 41 & 26 & 59 & 0 \end{bmatrix}$$

,τα διανύσματα προσφοράς και ζήτησης $a = [5 \quad 2 \quad 7 \quad 6]$ ***και*** $b = [2 \quad 3 \quad 7 \quad 2 \quad 1 \quad 5]$ ***αντίστοιχα για ένα*** 4×6 ***πρόβλημα Μεταφοράς. Να προσδιορισθεί το δέντρο ξεκινήματος Balinski και να υπολογισθούν τα στοιχεία*** $x_{ij}(T),\ (i,j) \in T$**,** $u_i(T),\ i = 1,...,m$**,** $v_j(T),\ j = 1,...,n$ ***και*** $s_{ij}(T),\ (i,j) \notin T$**.**

Λύση:

Έστω T το δέντρο Balinski. Αρχικά θέτουμε $u_1(T) = 0$ και $v_j(T) = c_{1j},\ j = 1,...,6$. Έτσι οι τιμές των μεταβλητών που προκύπτουν σύμφωνα με τον πίνακα των δεδομένων θα είναι: $u_1(T) = 0,\ v_1(T) = -5,\ v_2(T) = 49,\ v_3(T) = 28,\ v_4(T) = 20,\ v_5(T) = 47,\ v_6(T) = 43$. Συνεπώς, μέχρι τώρα το δέντρο που έχει σχηματισθεί θα είναι το παρακάτω:



Εικόνα 3.0 – Αρχικό στάδιο κατασκευής του δέντρου Balinski

Για όλα τα τόξα (i, j) του παραπάνω δέντρου θα έχουμε $s_{ij}(T) = c_{ij} - u_i(T) - v_j(T) = 0$. Για τον υπολογισμό των άλλων βασικών τόξων του δέντρου αφαιρούμε την τιμή της δυϊκής

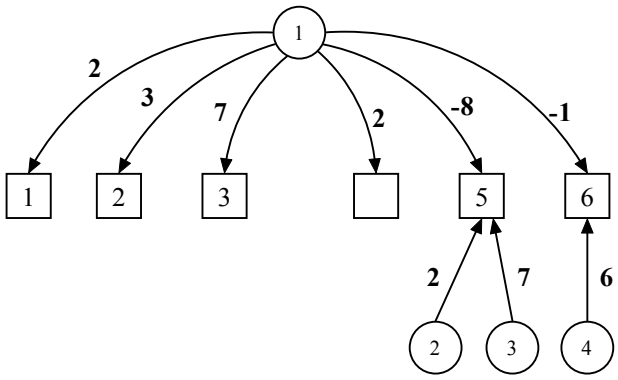
μεταβλητής v_j από όλα τα στοιχεία της στήλης j του πίνακα κόστους. Έτσι προκύπτει ο πίνακας

$$D=\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 24 & 13 & 4 & 41 & -47 & -30 \\ 3 & 14 & -14 & 26 & -19 & 10 \\ 6 & 1 & 13 & 6 & 12 & -43 \end{bmatrix}$$

Συνεπώς τα νέα βασικά τόξα $(i, j(i))$ του δέντρου Balinski θα είναι αυτά που ικανοποιούν τη σχέση

$$c_{ij(i)} - v_{j(i)}(T) = \min\left\{D^i{}_j : j = 1,...,n\right\} \quad , i = 2,...,m$$

όπου D^i η i γραμμή του πίνακα D. Με αυτόν τον τρόπο διαπιστώνουμε ότι τα νέα βασικά τόξα του δέντρου Balinski θα είναι τα τόξα (2,5), (3,5) και (4,6).Επίσης, οι δυϊκές μεταβλητές των υπολοίπων κόμβων γραμμών θα είναι $u_2 = -47, u_3 = -19$ και $u_4 = -43$.Συνεπώς τα διανύσματα των δυϊκών μεταβλητών θα είναι $u = \begin{bmatrix} 0 & -47 & -19 & -43 \end{bmatrix}$ και $v = \begin{bmatrix} -5 & 49 & 28 & 20 & 47 & 43 \end{bmatrix}$. Θέτουμε επίσης $x_{25} = 2$, $x_{35} = 7$ και $x_{46} = 6$, $x_{11} = 2, x_{12} = 3, x_{13} = 7, x_{14} = 2$ σύμφωνα με τα διανύσματα προσφοράς και ζήτησης. Επίσης θέτουμε $x_{15} = b(5) - x_{35} - x_{25} = 1 - 7 - 2 = -8$ και στη συνέχεια $x_{16} = b(6) - x_{46} = 5 - 6 = -1$. Εύκολα τώρα παράγουμε τον πίνακα των μειωμένων κοστών ο οποίος περιέχει και τις μεταβλητές απόφασης στα σκιασμένα κελιά. Το δέντρο Balinski και ο αντίστοιχος πίνακας φαίνονται στο παρακάτω σχήμα:



2	3	7		-8	-1
71	60	51		**2**	17
22	33	5		**7**	29
49	44	56		55	**6**

Εικόνα 3.1 – Το δέντρο Balinski και ο αρχικός πίνακας του προβλήματος.

3.1.2 Περιγραφή του Αλγορίθμου

Ο αλγόριθμος Παπαρρίζου ξεκινά με το δέντρο Balinski. Έστω T το τρέχον δέντρο. Αν ισχύει $x_{1j}(T) \geq 0$ για κάθε βασικό τόξο $(1, j)$ του δέντρου T, ο αλγόριθμος σταματάει

και T είναι το βέλτιστο δέντρο. Διαφορετικά προσδιορίζονται τα τόξα $(1, j)$ τέτοια ώστε $x_{1j}(T) < 0$. Σχηματίζουμε λοιπόν το σύνολο J ως εξής:

$$J = \left\{ j : \ j \in D \wedge x_{1j} < 0 \right\}$$

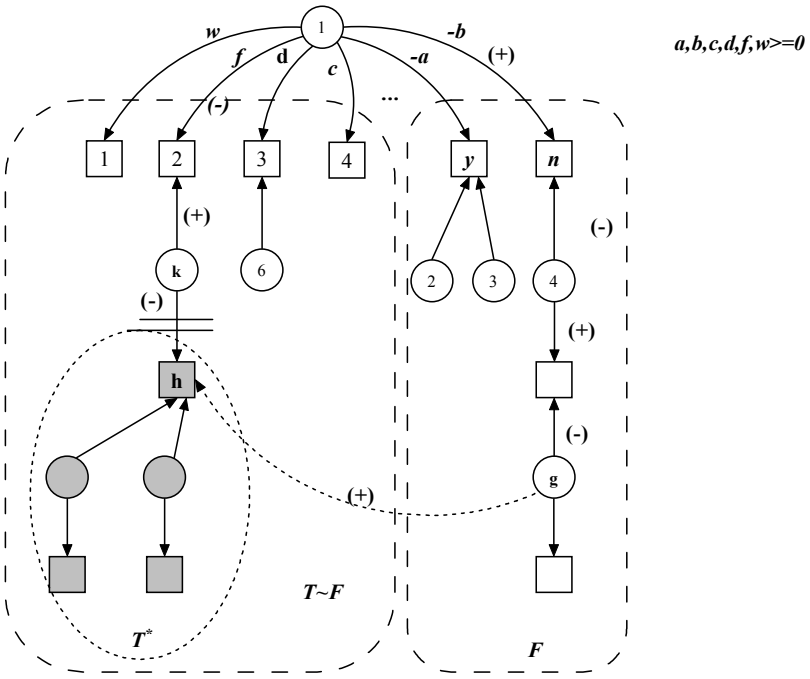
, όπου D το σύνολο των κόμβων ζήτησης του προβλήματος. Έστω τώρα T_j το υπόδεντρο του T που κόβεται από τη ρίζα, όταν διαγράφεται το τόξο $(1, j) \in T$. Συμβολίζουμε με F το δάσος που αποτελείται από δέντρα T_j τέτοια ώστε $j \in J$. Το εισερχόμενο τόξο (g, h) προσδιορίζεται από την παρακάτω σχέση:

$$\delta = s_{gh} = \min \left\{ s_{ij}(T) : i \in F, j \in T \sim F \right\} \geq 0$$

Το εισερχόμενο τόξο ως γνωστόν θα δημιουργήσει ένα διακριτό κύκλο $C = T \cup (g, h)$. Ο κύκλος αυτός μπορεί να αποδειχθεί ότι θα περιέχει πάντα τη ρίζα και ένα τόξο της μορφής $(1, f) \in T$ τέτοιο ώστε $f \in J$. Υπολογίζουμε τα σύνολα C^+ και C^- που περιέχουν τα τόξα εκείνα που ανήκουν στον κύκλο C και έχουν ίδια και αντίθετη φορά με το εισερχόμενο τόξο αντίστοιχα. Το εξερχόμενο τόξο (k, ℓ) επιλέγεται σύμφωνα με τη σχέση:

$$\varepsilon = x_{k\ell} = \min \left\{ -x_{1f}(T), \min \left\{ x_{ij}(T) \ , (i, j) \in C^- \right\} \right\} \geq 0$$

Στη συνέχεια, βλέπουμε μια γενική επανάληψη που εκτελεί ο αλγόριθμος. Παρατηρείστε τα σύνολα $F, T \sim F, T^*$. Οι αριθμοί δίπλα στα τόξα συμβολίζουν τις τιμές x_{ij} **.**



Εικόνα 3.2 – Μια γενική επανάληψη του αλγορίθμου

Ο κανόνας του Cunningham**[Cun]** εφαρμόζεται όπως ακριβώς και στο πρόβλημα Μεταφοράς χρησιμοποιώντας τον αλγόριθμο Simplex με τη διαφορά ότι υπολογίζουμε τα

επιλέξιμα τόξα από το σύνολο $C' = C^- \cup (1, f)$. Συνεπώς το σύνολο των επιλέξιμων τόξων σε αυτήν την περίπτωση θα είναι το

$$C'' = \left\{(i, j) \in C' : \ \left|x_{ij}(T)\right| = \varepsilon\right\}$$

Το παραπάνω σύνολο περιέχει υποψήφια εξερχόμενα τόξα. Το εξερχόμενο τόξο (k, ℓ) θα είναι το πρώτο επιλέξιμο τόξο που συναντάμε όταν διαγράφουμε τον κύκλο $C = T \cup (g, h)$ ξεκινώντας από τη ρίζα με τη φορά του εισερχόμενου τόξου.

Όπως και στον αλγόριθμο Simplex, συμβολίζουμε με T^* το υπόδεντρο που κόβεται από τη ρίζα όταν διαγραφεί το τόξο $(k, \ell) \in T$. Το νέο δέντρο T' είναι το δέντρο $T \cup (g, h) \sim (k, \ell)$. Η διαδικασία επαναλαμβάνεται και ο αλγόριθμος τερματίζει όταν $F = \varnothing$, δηλαδή όταν δεν θα υπάρχουν τόξα $(1, j) \in T$ τέτοια ώστε $x_{1j}(T) < 0$.

Τα σύνολα F και $T \sim F$ ανανεώνονται εύκολα και συγκεκριμένα αν το υπόδεντρο T^* κρεμαστεί μέσω του εισερχόμενου τόξου (g, h) από ένα υπόδεντρο του δάσους F, θέτουμε $F = F \cup (g, h) \cup T^*$ και $T \sim F = (T \sim F) \sim T$, αλλιώς θέτουμε $F = F \sim T^*$ και $T \sim F = (T \sim F) \cup (g, h) \cup T^*$. Επίσης, ανανεώνονται τα μειωμένα κόστη $s_{ij}(T)$ των τόξων (i, j) των οποίων μόνο ένα άκρο $u \in T^*$. Τέλος, η ανανέωση των μεταβλητών $x_{ij}(T)$ γίνεται με τον γνωστό τρόπο:

$$x_{ij}(T') = \begin{cases} x_{ij}(T) + \varepsilon & ,\forall (i, j) \in C^+ \\ x_{ij}(T) - \varepsilon & ,\forall (i, j) \in C^- \\ x_{ij}(T) & ,\forall (i, j) \notin C \end{cases}$$

3.1.3 Βηματική Περιγραφή του Αλγορίθμου

ΒΗΜΑ 0:*(Καθορισμός αρχικών μεταβλητών)*
Υπολόγισε το δέντρο Balinski T με τη μέθοδο που περιγράψαμε και καθόρισε το σύνολο F θέτοντας $i \in F$ αν και μόνον αν $i \in T_j$ τέτοιο ώστε $x_{1j} < 0$.

ΒΗΜΑ 1:*(Έλεγχος βελτιστότητας)*

βέλτιστο και ο αλγόριθμος τερματίζει, αλλιώς πήγαινε στο βήμα 2.

ΒΗΜΑ 2:*(Επιλογή εισερχομένου τόξου)*
Υπολόγισε το εισερχόμενο τόξο (g, h) ελέγχοντας τα μειωμένα κόστη των μη βασικών τόξων με βάση τον τύπο $\delta = s_{gh} = \min\left\{s_{ij}(T) : i \in F, j \in T \sim F\right\}$.

ΒΗΜΑ 3:*(Επιλογή εξερχόμενου τόξου)*
Υπολόγισε τα σύνολα τόξων C^+, C^- και καθόρισε τη μεταβλητή ε, θέτοντας $\varepsilon = \min\left\{-x_{1f,} \min\left\{x_{ij}(T) : (i, j) \in C^-\right\}\right\}$.Βρες το σύνολο C'' των επιλέξιμων τόξων, θέτοντας $C'' = \left\{(i, j) \in C^- \cup (1, f) : \ \left|x_{ij}(T)\right| = \varepsilon\right\}$. Επέλεξε το εξερχόμενο τόξο $(k, \ell) \in C^- \cup (1, f)$ με τον τροποποιημένο κανόνα του Cunninghum, δηλαδή το εξερχόμενο τόξο θα είναι το πρώτο

επιλέξιμο τόξο $(r,t)\in C^{-}\cup(1,f)$ που συναντάμε κατά την διάσχιση του κύκλου C, ξεκινώντας από τη ρίζα του δέντρου.

ΒΗΜΑ 4:*(Ανανέωση των μεταβλητών)*

κάθε τόξο (i,j) .Υπολόγισε το δέντρο $T*$ που είναι το δέντρο που αποκόβεται από τη ρίζα όταν αφαιρείται το εξερχόμενο τόξο. Αν $h\in T*$ θέσε $q=-\delta>0$, αλλιώς θέσε $q=\delta<0$. Στη συνέχεια εάν ο κόμβος $b\in T*$ είναι κόμβος-γραμμή θέσε $s_{bj}(T')=s_{bj}(T)-$ αλλιώς θέσε $s_{ib}(T')=s_{ib}(T)+q$, $i=1,...,m$.

ΒΗΜΑ 5:*(Ανανέωση του τρέχοντος δέντρου)*

Στη θα παρουσιάσουμε την ακριβή επίλυση ενός παραδείγματος.

Παράδειγμα 3.2

Δίνονται το διάνυσμα προσφοράς $a=\begin{bmatrix}5 & 2 & 7 & 6\end{bmatrix}$***, το διάνυσμα ζήτησης*** $b=\begin{bmatrix}2 & 3 & 7 & 2 & 1 & 5\end{bmatrix}$***, και ο πίνακας κόστους***

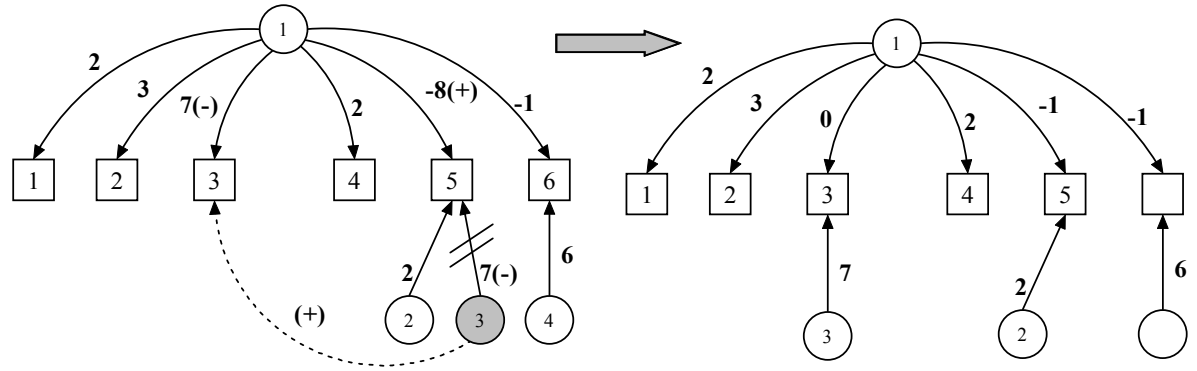
$$C=\begin{bmatrix}-5 & 49 & 28 & 20 & 47 & 43\\ 19 & 62 & 32 & 61 & 0 & 13\\ -2 & 63 & 14 & 46 & 28 & 53\\ 1 & 50 & 41 & 26 & 59 & 0\end{bmatrix}$$

για ένα 4×6 ***πρόβλημα Μεταφοράς. Να λυθεί με τον αλγόριθμο Παπαρρίζου και να χρησιμοποιηθεί το δέντρο Balinski ως εφικτό δέντρο ξεκινήματος.***

Λύση:

Αρχικά, πρέπει να προσδιορίσουμε το δέντρο ξεκινήματος Balinski(σχήμα 3.1).

Επανάληψη 1



2	3	7	2	-8	-1
71	60	51	88	**2**	17
22	33	5	45	**7**	29
49	44	56	49	55	**6**

-5

2	3	0	2	-1	-1
71	60	51	88	**2**	17
17	28	**7**	40	-5	24
49	44	56	49	55	**6**

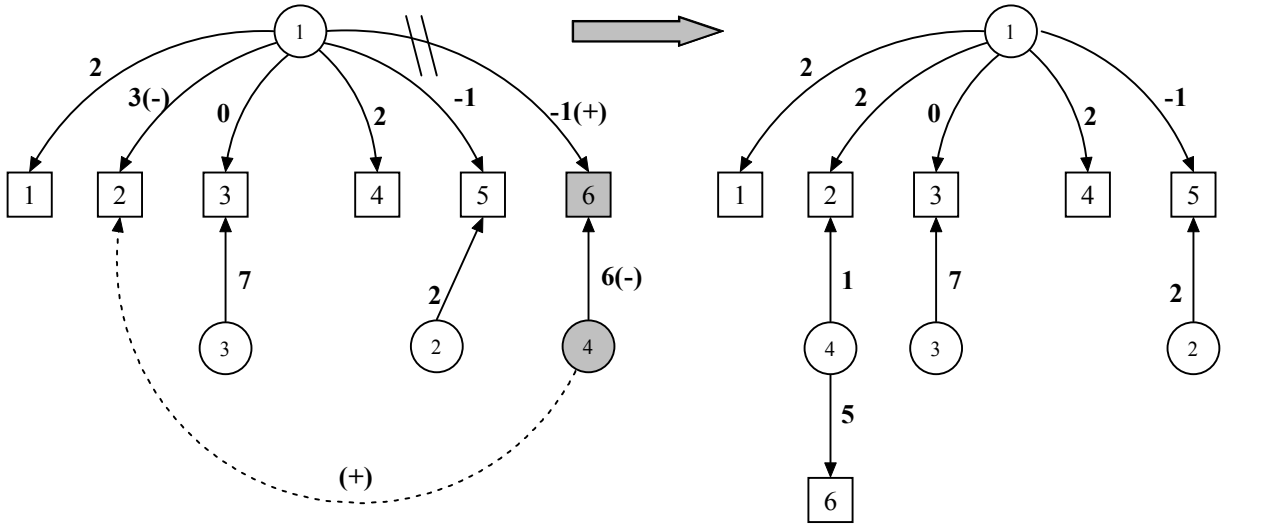
Εικόνα 3.3 – Η πρώτη επανάληψη του αλγορίθμου

- Είναι $F=\{2,3,4\}$, $T\sim F=\{1,2,3,4\}$, $\delta=\min\{s_{ij}(T):i\in F,j\in T\sim F\}=s_{33}=5$, συνεπώς $(g,h)=(3,3)$

- Τα σύνολα C^+, C^-, φαίνονται στο σχήμα 3.3. Χρησιμοποιώντας τη γνωστή σχέση $\varepsilon = \min\left\{-x_{1f,} \min\left\{x_{ij}(T) : (i,j) \in C^-\right\}\right\}$, προσδιορίζουμε το εξερχόμενο τόξο (k,ℓ). Έτσι $\varepsilon = 7$ και $(k,\ell) = (3,5)$.
- Η αναλυτική ανανέωση των μεταβλητών x_{ij}, s_{ij} φαίνεται στο σχήμα 3.3

Επανάληψη 2

- Είναι $F = \{2,4\}$, $T \sim F = \{1,2,3,4\}$, $\delta = \min\left\{s_{ij}(T) : i \in F, j \in T \sim F\right\} = s_{42} = 60$, συνεπώς $(g,h) = (4,2)$
- Τα σύνολα C^+, C^-, φαίνονται στο σχήμα 3.4. Χρησιμοποιώντας τη γνωστή σχέση $\varepsilon = \min\left\{-x_{1f,} \min\left\{x_{ij}(T) : (i,j) \in C^-\right\}\right\}$, προσδιορίζουμε το εξερχόμενο τόξο (k,ℓ). Έτσι $\varepsilon = 1$ και $(k,\ell) = (1,6)$.
- Η αναλυτική ανανέωση των μεταβλητών x_{ij}, s_{ij} φαίνεται στο σχήμα 3.4



2	3	0	2	-1	-1	
71	60	51	88	**2**	17	
17	28	**7**	40	-5	24	
49	44	56	49	55	**6**	**-44**
						+44

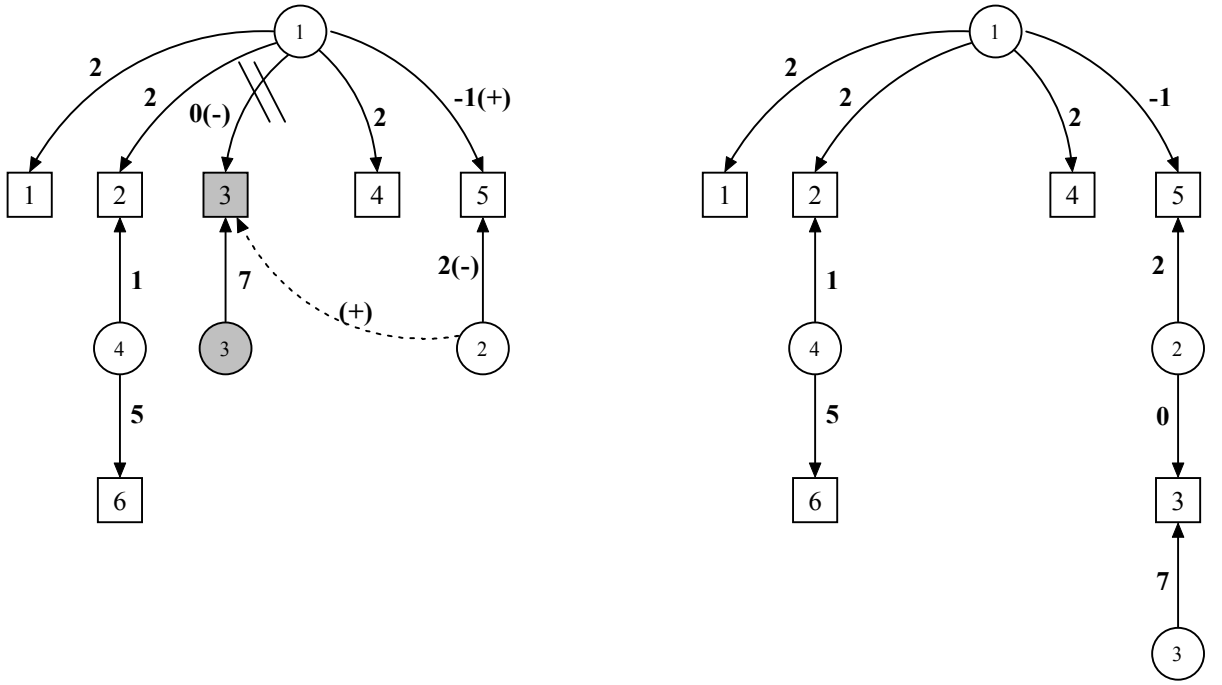
2	2	0	2	-1	44
71	60	51	88	**2**	61
17	28	**7**	40	-5	68
5	**1**	12	5	11	**5**

Εικόνα 3.4 – Η δεύτερη επανάληψη του αλγορίθμου

Επανάληψη 3

- Είναι $F = \{2\}$, $T \sim F = \{1,2,3,4,6\}$, $\delta = \min\left\{s_{ij}(T) : i \in F, j \in T \sim F\right\} = s_{23} = 51$, συνεπώς $(g,h) = (2,3)$.
- Τα σύνολα C^+, C^-, φαίνονται στο σχήμα 3.5. Χρησιμοποιώντας τη γνωστή σχέση $\varepsilon = \min\left\{-x_{1f,} \min\left\{x_{ij}(T) : (i,j) \in C^-\right\}\right\}$, προσδιορίζουμε το εξερχόμενο τόξο (k,ℓ). Έτσι $\varepsilon = 0$ και $(k,\ell) = (1,3)$.
- Η αναλυτική ανανέωση των μεταβλητών x_{ij}, s_{ij} φαίνεται στο σχήμα 3.5.

Παρατηρείστε ότι επειδή $\varepsilon = 0$, οι μεταβλητές απόφασης του δέντρου παραμένουν αμετάβλητες. Μια επανάληψη αυτού του είδους ονομάζεται εκφυλισμένη(*degenerate*).



2	2	0	2	-1	44
71	60	51	88	2	61
17	28	7	40	-5	68
5	1	12	5	11	5

+51

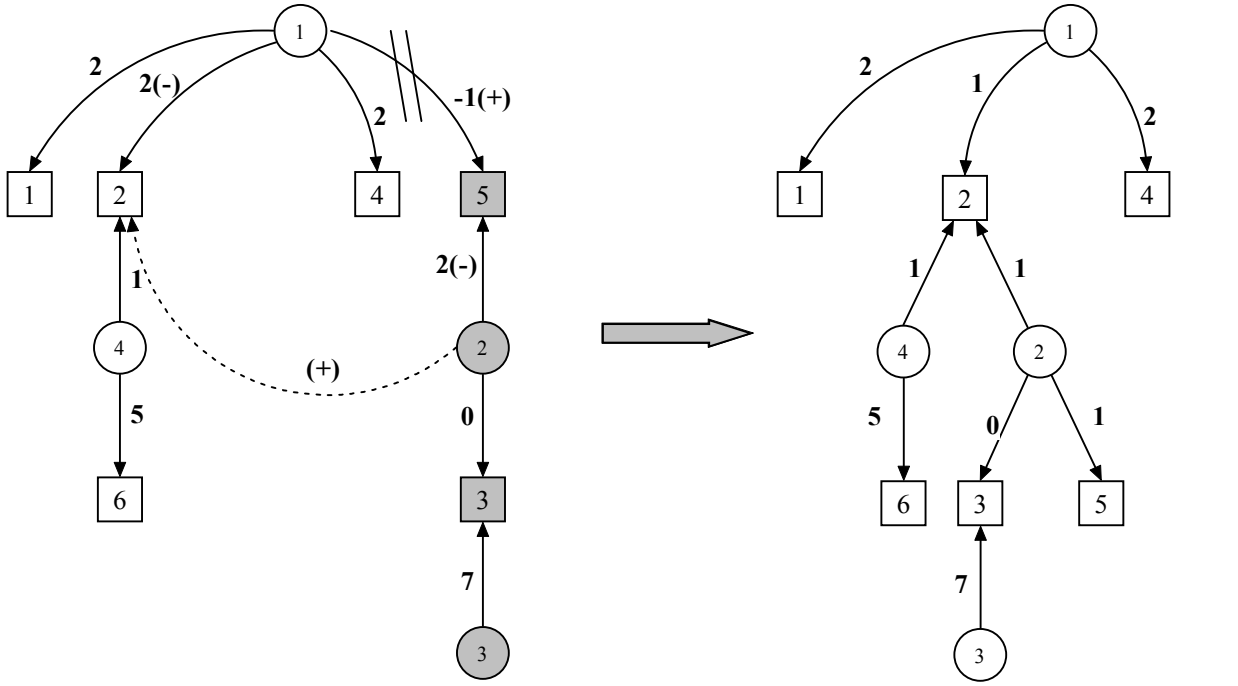
-51

2	2	-51	2	-1	44
71	60	0	88	2	61
68	79	7	91	46	119
5	1	-39	5	11	5

Εικόνα 3.5 - Η τρίτη επανάληψη του αλγορίθμου

Επανάληψη 4

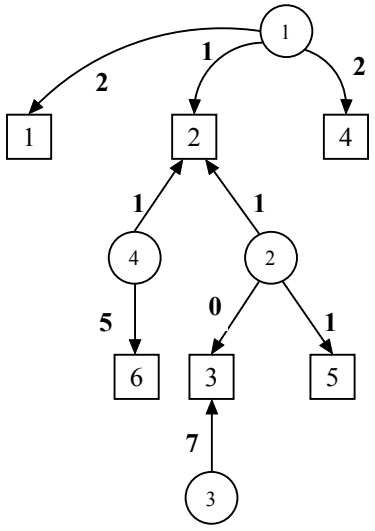
- Είναι $F=\{2,3\}$, $T\sim F=\{1,2,4,6\}$, $\delta=\min\{s_{ij}(T): i\in F, j\in T\sim F\}=s_{22}=60$, συνεπώς $(g,h)=(2,2)$.
- Υπολογίζονται τα σύνολα C^+, C^-. Συνεπώς θα έχουμε $C^+=\{(2,2),(1,5)\}$, $C^-=\{(1,2),(2,5)\}$.Έχοντας προσδιορίσει τα σύνολα των τόξων, χρησιμοποιούμε τώρα τη γνωστή σχέση $\varepsilon=\min\{-x_{1f,}\min\{x_{ij}(T):(i,j)\in C^-\}\}$ για να προσδιορίζουμε το εξερχόμενο τόξο (k,ℓ). Έτσι $\varepsilon=1$ και $(k,\ell)=(1,5)$.
- Η αναλυτική ανανέωση των μεταβλητών x_{ij}, s_{ij} φαίνεται στο σχήμα 3.6. Ανανεώνονται μόνο οι τιμές μειωμένων κοστών των τόξων που έχουν έναν μόνο κόμβο στο δέντρο T^*. Συγκεκριμένα, προσθέτουμε +60 στις γραμμές 3, 5 του πίνακα s_{ij} ενώ προσθέτουμε -60 στις γραμμές 2, 3 του πίνακα s_{ij}. Η ανανέωση αυτή γίνεται με τη μέθοδο που αναφέρθηκε στο βήμα 4 της περιγραφής του αλγορίθμου. Μπορούμε όμως εύκολα να καταλάβουμε που πρέπει να προσθέσουμε και αντίστοιχα να αφαιρέσουμε δ, αν λάβουμε υπ' όψιν το γεγονός ότι το μειωμένο κόστος του εισερχόμενου τόξου (g,h) πρέπει να μηδενιστεί μετά το πέρας της επανάληψης, αφού το μειωμένο κόστος κάθε βασικού τόξου είναι πάντα ίσο με το μηδέν. Συνεπώς θα πρέπει να αφαιρέσουμε δ από τη δεύτερη γραμμή του πίνακα, επειδή το εισερχόμενο τόξο είναι το τόξο $(2,2)$, πράγμα που σημαίνει ότι θα πρέπει να προσθέσουμε την ποσότητα δ από τις στήλες του πίνακα s_{ij}.



2	2	-51	2	-1	44	
71	60	**0**	88	**2**	61	-60
68	79	**7**	91	46	119	-60
5	**1**	-39	5	11	**5**	
		+60		**+60**		

2	1	9	2	60	44
11	**1**	**0**	28	**1**	1
8	19	**7**	31	46	59
5	**1**	21	5	71	**5**

Εικόνα 3.6 – Η τέταρτη επανάληψη του αλγορίθμου



Εικόνα 3.7 – Το τελικό βέλτιστο δέντρο.

Επειδή στο τελευταίο δέντρο που παράχθηκε, παρατηρούμε ότι $x_{1j} \geq 0 \ \forall (i,j) \in T$, συμπεραίνουμε ότι $F = \varnothing$. Άρα ο αλγόριθμος τερματίζει και το δέντρο που παράχθηκε είναι βέλτιστο.

3.1.4 Προγραμματισμός του Αλγορίθμου

Για τον προγραμματισμό του αλγορίθμου καθώς και των αλγορίθμων που θα περιγραφούν στη συνέχεια χρησιμοποιήθηκαν παρόμοιες τεχνικές προγραμματισμού με

αυτές που χρησιμοποιήθηκαν και στον αλγόριθμο Simplex. Συνεπώς, δε χρειάζεται να αναλύσουμε ξανά τις δομές δεδομένων που χρησιμοποιήθηκαν. Ο ψευδοκώδικας του αλγορίθμου σε μορφή κλήσεως συναρτήσεων φαίνεται παρακάτω:

Αλγόριθμος T2
Ο παρακάτω αλγόριθμος λύνει το ισοζυγισμένο πρόβλημα μεταφοράς , χρησιμοποιώντας το δενδρικό αλγόριθμο Παπαρρίζου με ξεκίνημα το δέντρο Balinski.
Είσοδος: *Το διάνυσμα προσφοράς* **A(mx1)**,*το διάνυσμα ζήτησης* **B(nx1)** *και ο πίνακας κόστους* **C(mxn)**.
Έξοδος : *Η λύση του προβλήματος Μεταφοράς* **X(mxn)**.

```
1.  [X,S]=BalinskiTree(A,B,C,m,n);
2.  p=InitializeNodeFather(X,m,n);
3.  t=InitializeNodeDirection(m,n);
4.  d=InitializeNodeDepth(p,m,n);
5.  xv=Vectorx(p,t,X,m,n);
6.  stopalgorithm=0;
7.  while stopalgorithm~=1
8.    [F,Fbar]=ForestsF(xv,p,m,n);
9.    [min,g,h,goon]=EnteringArc(S,F,Fbar,m,n);
10.   [e,k,l,Cplus,Cminus]=LeavingArc(g,h,p,m,n,d,t,xv,goon);
11.   [e1,e2,f1,z,Tcut]=Steam(p,g,h,k,l,d,m,n);
12.   [S,X]=UpdateVariables(min,e,g,h,k,l,Tcut,Cminus,Cplus,m,n,t,p,S,X);
13.   d=UpdateNodeDepth(p,m,n,Tcut,z,d,e1,e2,f1);
14.   p=UpdateNodeFather(p,z,e2,f1);
15.   xv=Vectorx(p,t,X,m,n);
16.   fw=find(X(1,:)>=0);
17.     if length(fw)==n
18.       stopalgorithm=1;
19.     end
20. end
```

Αλγόριθμος 3.1.1 – Ο κώδικας υλοποίησης του δενδρικού αλγόριθμου Παπαρρίζου με ξεκίνημα το δέντρο Balinski

Παραπάνω βλέπουμε όλες τις συναρτήσεις που χρησιμοποιήσαμε για τον προγραμματισμό του αλγορίθμου. Υλοποιώντας αυτές τις συναρτήσεις σωστά σε οποιαδήποτε γλώσσα προγραμματισμού, μπορεί εύκολα κάποιος να προγραμματίσει τον αλγόριθμο με ανάλογο τρόπο. Στον παραπάνω κώδικα, μπορούμε να δούμε στις τελευταίες γραμμές(γραμμές 16-20) τον τρόπο που επιτυγχάνεται ο έλεγχος τερματισμού. Στο τέλος λοιπόν κάθε επανάληψης ελέγχονται οι τιμές των μεταβλητών απόφασης των τόξων $(1, j) \in T$. Αν είναι όλες θετικές, τότε το σύνολο F είναι το κενό σύνολο, τίθεται μια μεταβλητή τύπου flag ίση με τη μονάδα (**`stopalgorithm=1`**) και ο αλγόριθμος σταματάει, έχοντας προσδιορίσει το βέλτιστο δέντρο. Σε διαφορετική περίπτωση, δηλαδή αν υπάρχει τουλάχιστον ένα τόξο $(1, j) \in T$ τέτοιο ώστε $x_{1j} < 0$, τότε ο αλγόριθμος συνεχίζει την εκτέλεσή του θέτοντας τη σημαία **`stopalgorithm`** ίση με το μηδέν. Βέβαια εδώ πρέπει να πούμε ότι ο τρόπος προγραμματισμού που χρησιμοποιήσαμε δεν είναι ο μοναδικός. Ο αλγόριθμος μπορεί να προγραμματιστεί και με άλλες τεχνικές. Εμείς προγραμματίσαμε με στόχο την επίτευξη καλών υπολογιστικών επιδόσεων. Έτσι χρησιμοποιήσαμε συναρτήσεις που δεν ανανεώνουν εξαρχής ορισμένες δομές, όπως το βάθος των κόμβων και το διάνυσμα πατέρα-κόμβου, άλλα βασίζονται σε μερικά θεωρητικά αποτελέσματα που παρουσιάσθηκαν στον αλγόριθμο Simplex. Στη συνέχεια, ακολουθεί η ανάλυση των σημαντικότερων κατά την άποψη μας συναρτήσεων.

- **[X , S] = BalinskiTree(A , B , C , m ,n)**

Αλγόριθμος BalinskiTree
Ο αλγόριθμος που προσδιορίζει του εφικτό δέντρο ξεκινήματος Balinski.
Είσοδος: *Το διάνυσμα προσφοράς* **A(mx1)**, *το διάνυσμα ζήτησης* **B(nx1)** *και ο πίνακας κόστους* **C(mxn)**.

Έξοδος : *Το εφικτό δέντρο* **X(mxn)** *και ο αντίστοιχος πίνακας μειωμένων κοστών* **S(mxn)** .

```
1.  function [X,S]=BalinskiTree(A,B,C)
2.  m=length(A);
3.  n=length(B);
4.  U(1)=0;
5.  for j=1:n
6.    V(j)=C(1,j);
7.  end
8.  X=inf;
9.  for i=2:m
10.   min=inf;
11.   for j=1:n
12.     X(1,j)=0;
13.     if C(i,j)-V(j)<min
14.       min= C(i,j)-V(j);
15.       j1=j;
16.     end
17.     U(i)=min;
18.   end
19.   X(i,j1)=0;
20. end
21. for i=1:m
22.   for j=1:n
23.     S(i,j)=C(i,j)-U(i)-V(j);
24.   end
25. end
26. for i=1:m
27.   for j=1:n
28.     if X(i,j)~=0
29.       X(i,j)=inf;
30.     end
31.   end
32. end
33. for j=1:n
34.   f=0;
35.   Sum=0;
36.   for i=2:m
37.     if X(i,j)~=inf
38.       f=f+1;
39.       if f==0
40.         X(1,j)=B(j);
41.       else
42.         X(i,j)=A(i);
43.         Sum=Sum+A(i);
44.       end
45.     end
46.   end
47.   X(1,j)=B(j)-Sum;
48. end
```

Αλγόριθμος 3.1.2 – Ο αλγόριθμος υπολογισμού του δέντρου ξεκινήματος Balinski.

Αν παρατηρήσουμε προσεχτικά τον παραπάνω κώδικα, θα παρατηρήσουμε ότι αρχικά υπολογίζονται οι δυϊκές μεταβλητές $v_j(T), j=1,...,n$ που αντιστοιχούν στους κόμβους – στήλες του δέντρου και στη συνέχεια υπολογίζονται αντίστοιχα οι δυϊκές μεταβλητές $u_i(T), i=1..m$ που αντιστοιχούν στους κόμβους γραμμές του δέντρου(γραμμές 4-18). Οι μεταβλητές $u_i(T), i=1..m$, υπολογίζονται με βάση τον τύπο που είδαμε στην περιγραφή του δέντρου Balinski:

$$u_i(T) = \min\left\{c_{ij} - v_j(T) : j = 1,...,n\right\} \quad , i = 2,...,m$$

Στη συνέχεια υπολογίζονται τα μειωμένα κόστη s_{ij} και τέλος υπολογίζονται οι μεταβλητές απόφασης του δέντρου(γραμμές 36 - 47).

- **[F , Fbar] =ForestsF(xv , p , m , n)**

Αλγόριθμος ForestsF
Ο αλγόριθμος που υπολογίζει τα δέντρα F και T~F.
Είσοδος: xv(m+n) , p(m+n) , *οι διαστάσεις του προβλήματος* **m , n**
Έξοδος : *Τα διανύσματα των κόμβων των δύο δέντρων* **F( ) , Fbar( )**

```
1.  function [F,Fbar]=ForestsF(xv,p,m,n)
2.  y1=0;
3.  y2=0;
4.  s1=0;
5.  s2=0;
6.  for j=1:m+n
7.    F(j)=0;
8.    F1(j)=0;
9.    Fbar(j)=0;
10.   F2(j)=0;
11. end
12. s=1;
13. for j=m+1:m+n
14.   if (p(j)==1)&(xv(j)<0)
15.     F1=Preorder(j,p,m,n);
16.     y1=length(F1)+1;
17.     F1(y1:m+n)=0;
18.     for i=s1+1:y1+s1+1
19.       F(i)=F1(i-s1);
20.     end
21.     s1=s1+y1-1;
22.   end
23. end
24. for j=m+1:m+n
25.   if (p(j)==1)&(xv(j)>=0)
26.     F2=Preorder(j,p,m,n);
27.     y2=length(F2)+1;
28.     F2(y2:m+n)=0;
29.     for i=s2+1:y2+s2+1
30.       Fbar(i)=F2(i-s2);
31.     end
32.     s2=s2+y2-1;
33.   end
34. end
```

Αλγόριθμος 3.1.3 – Ο αλγόριθμος υπολογισμού των δέντρων *F*, *T~F*.

Ο παραπάνω αλγόριθμος υπολογίζει τα δέντρα F και $T \sim F$, τα οποία χρησιμεύουν όπως είδαμε στον υπολογισμό του εισερχόμενου τόξου καθώς και στον καθορισμό του τερματισμού του αλγορίθμου. Η ιδέα πάνω στην οποία βασίζεται ο αλγόριθμος είναι η παρακάτω: Αρχικά, ελέγχει και βρίσκει τα τόξα εκείνα $(1, j)$ τέτοια ώστε $x_{1j} < 0$ και εκτελεί μια preorder διάσχιση του δέντρου T_j , όπου T_j είναι το δέντρο που έχει ρίζα τον κόμβο στήλη j. Στη συνέχεια αποθηκεύει όλους τους κόμβους που προκύπτουν από τις διασχίσεις αυτών των υποδέντρων σε ένα διάνυσμα **F**, το οποίο είναι το διάνυσμα που αποθηκεύει τους κόμβους του συνόλου F.

Για τον υπολογισμό του συνόλου $T \sim F$, θα μπορούσαμε να ελέγχουμε τους κόμβους του δέντρου και αναλόγως αν αυτοί ανήκουν στο ήδη υπολογισμένο σύνολο F, να τοποθετούνται ή όχι στο σύνολο $T \sim F$. Ακολουθούμε όμως μια διαδικασία παρόμοια με

την προηγούμενη, δηλαδή διασχίζουμε τα δέντρα T_j με $x_{1j} \geq 0$ έτσι ώστε να αποφύγουμε το πέρασμα όλων των κόμβων, χρησιμοποιώντας έτσι την αποδοτική αναδρομική διαδικασία **`Preorder()`**, όπως αυτή περιγράφηκε και στον αλγόριθμο Simplex. Οι κόμβοι του συνόλου $T \sim F$ αποθηκεύονται στο διάνυσμα **`Fbar`**.

- **[e, k, l, Cplus, Cminus] = LeavingArc(g, h, p, m, n, d, t, xv, goon)**

Αλγόριθμος LeavingArc
Ο αλγόριθμος που υπολογίζει το εξερχόμενο τόξο με βάση τον κανόνα του Cunningham .
Είσοδος: g , h, p(m+n), m , n, d(m+n), t(m+n), xv(m+n)[1], goon[2]
Έξοδος : *Η μεταβλητή εξερχόμενου τόξου* **e,** *τα άκρα του εξερχόμενου τόξου* **k, l ,** *τα σύνολα* **Cplus, Cminus**

```
1.  function [e,k,l,Cplus,Cminus]=LeavingArc(g,h,p,m,n,d,t,xv,goon)
2.  g1=g;
3.  h1=h;
4.  joint=1;
5.  for i=1:m+n
6.    Cplus(i)=0;
7.    Cminus(i)=0;
8.  end
9.  s=g1;
10. i=1;
11.  if (goon==1)
12.   Cirg(1)=g1;
13.   while (s~=1)
14.     i=i+1;
15.     s=p(s);
16.     Cirg(i)=s;
17.   end
18.   counter=i;
19.   s=h1+m;
20.   j=1;
21.   Cirh(1)=h1+m;
22.   while (s~=1)
23.     j=j+1;
24.     s=p(s);
25.     Cirh(j)=s;
26.   end
27.   joint=1;
28.   gm=0;
29.   hp=0;
30.   for q=1:i-1
31.     Circ(q)=Cirg(q);
32.     if t(Cirg(q))~=0
33.       gm=gm+1;
34.       Cminus(gm)=Cirg(q);
35.     else
36.       hp=hp+1;
37.       Cplus(hp)=Cirg(q);
38.    end
39.   end
40.   for q=1:j-1
41.     Circ(i-1+q)=Cirh(q);
42.     if t(Cirh(q))~=0
43.       hp=hp+1;
44.       Cplus(hp)=Cirh(q);
45.     else
```

[1] Το διάνυσμα *xv* είναι ένα διάνυσμα που ορίζεται ως εξής $xv(i) = X(i, p(i)),\ i = 1,...,m+n,\ i \neq root$

[2] Η μεταβλητή goon καθορίζει το αν θα συνεχιστεί η εκτέλεση του αλγορίθμου ή όχι.(Επιστρέφεται ως έξοδος από τη συνάρτηση **`EnteringArc`**). Είναι πάντα 1 εκτός από την περίπτωση που το πρόβλημα είναι εξ' αρχής βέλτιστο.

```
46.       gm=gm+1;
47.       Cminus(gm)=Cirh(q);
48.     end
49.   end
50.   Circ(i-2+j+1)=joint;
51.   epos=inf;
52.   for i=1:gm
53.     if xv(Cminus(i))<epos
54.       epos=xv(Cminus(i));
55.     end
56.   end
57.   done=0;
58.   q=0;
59.   for i=1:hp
60.     if xv(Cplus(i))<0
61.       q=q+1;
62.       enegarc(q)=abs(xv(Cplus(i)));
63.       oppositearc(q)=Cplus(i);
64.       done=1;
65.     end
66.   end
67.   for i=1:q
68.     for j=1:counter
69.       if oppositearc(q)==Cirg(j)
70.         opposite=oppositearc(q);
71.         eneg=enegarc(q);
72.       end
73.     end
74.   end
75.   if epos<=eneg
76.     e=epos;
77.   else
78.     e=eneg;
79.   end
80.   x=0;
81.   for i=1:gm
82.     if xv(Cminus(i))==e
83.       x=x+1;
84.       admis(x)=Cminus(i);
85.     end
86.   end
87.   if (done==1)&(abs(xv(opposite))==e)
88.     k=1;
89.     l=opposite-m;
90.   else
91.     node=g;
92.     stop=0;
93.     while node~=joint
94.       for count=1:x
95.         if node==admis(count)
96.           wanted=count;
97.           stop=1;
98.         end
99.       end
100.     node=p(node);
101.    end
102.    if stop==0
103.      node=h+m;
104.      while stop==0
105.      for count=1:x
106.        if node==admis(count)
107.          wanted=count;
108.          stop=1;

109.        end
110.      end
111.    node=p(node);
112.    end
```

```
113.  end
114.  if admis(wanted)>m
115.    k=p(admis(wanted));
116.    l=admis(wanted)-m;
117.  else
118.    l=p(admis(wanted))-m;
119.    k=admis(wanted);
120.  end
121.  end
122.  end
```

Αλγόριθμος 3.1.4 – Ο αλγόριθμος υπολογισμού του εξερχόμενου τόξου με βάση τον κανόνα του Cunningham.

Στον παραπάνω κώδικα γίνεται ο υπολογισμός του εξερχόμενου τόξου (k,ℓ) και των συνόλων των τόξων C^+, C^-. Επίσης, ελέγχεται αν ο κύκλος $C = C^- \cup (1,f)$ περιέχει περισσότερα από ένα επιτρεπτά τόξα (*admissible arcs*) και στη συνέχεια εφαρμόζεται ο κανόνας του Cunningham για τον καθορισμό του εξερχόμενου τόξου.

Συγκεκριμένα, στις γραμμές 12-17 του κώδικα υπολογίζονται οι κόμβοι που ανήκουν στο δρόμο από τον κόμβο g προς τη ρίζα του δέντρου. Στη συνέχεια(γραμμές 21-26), υπολογίζονται οι κόμβοι που ανήκουν στο δρόμο από τον κόμβο h στη ρίζα του δέντρου. Οι κόμβοι αυτοί αποθηκεύονται στα διανύσματα **Cirg** και **Cirh** αντίστοιχα. Στις επόμενες γραμμές του κώδικά υπολογίζονται με βάση τα προηγούμενα διανύσματα ο κύκλος C και τα σύνολα C^+, C^- τα οποία αποθηκεύονται στα διανύσματα **Cplus** και **Cminus** ως εξής: Αν **Cplus[i]== k** τότε στο σύνολο C^+ περιέχεται ή το τόξο $(k, p(k))$ ή το τόξο $(p(k), k)$ ανάλογα με την τιμή του διανύσματος t στη συγκεκριμένη θέση, όπου t το διάνυσμα που ορίζεται ως εξής:

$$t(i) = \begin{cases} 0\ , & (p(i), i) \in T \\ 1\ , & (i, p(i)) \in T \end{cases}$$

Αφού προσδιορισθεί το C^-, ο αλγόριθμος ψάχνει να βρει τα τόξα εκείνα των οποίων η μεταβλητή απόφασης είναι η ελάχιστη των μεταβλητών απόφασης των τόξων που ανήκουν στο σύνολο C^-. Τα τόξα αυτά είναι τα επιλέξιμα τόξα και αποθηκεύονται στο διάνυσμα **admis**. Επίσης, ελέγχεται αν είναι επιλέξιμο και το τόξο $(1,f)$, δηλαδή αν ισχύει:

$$\left| x_{1f} \right| = \min \left\{ x_{ij}(T) : (i,j) \in C^- \right\}$$

Αν ισχύει η παραπάνω σχέση, τότε επιλέξιμο γίνεται και το τόξο $(1,f)$, δηλαδή το τόξο που ανήκει στον κύκλο C και έχει αρνητική μεταβλητή απόφασης. Στη συνέχεια, διαγράφουμε τον κύκλο κατά τη φορά του εισερχόμενου τόξου (93-113) και υπολογίζεται τελικά το εξερχόμενο τόξο που αποθηκεύεται στη μεταβλητή **wanted**. Συνεπώς, το εξερχόμενο τόξο θα είναι ανάλογα ή το τόξο **(wanted,p(wanted))** ή το τόξο **(p(wanted),wanted)** ανάλογα με το αν ο κόμβος **admis(wanted)** είναι κόμβος γραμμή ή κόμβος στήλη. Αυτό φαίνεται αναλυτικά στις γραμμές του κώδικα 114-120. Αυτή η συνάρτηση είναι μια συνάρτηση η οποία χρησιμοποιήθηκε με μερικές μετατροπές και στην εφαρμογή του αλγορίθμου Simplex στο πρόβλημα Μεταφοράς.

Οι άλλες συναρτήσεις που χρησιμοποιεί ο αλγόριθμος είναι παρόμοιες με αυτές που χρησιμοποιήθηκαν από τον αλγόριθμο Simplex για το πρόβλημα Μεταφοράς και συνεπώς δεν θα αναλυθούν.

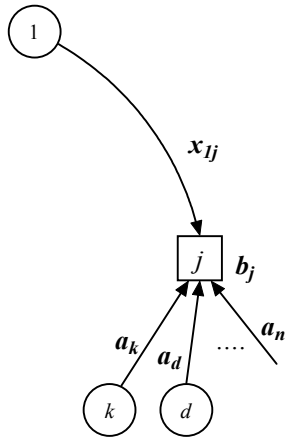
3.2 ΕΦΑΡΜΟΓΗ ΣΤΟ ΠΡΟΒΛΗΜΑ ΑΝΤΙΣΤΟΙΧΗΣΗΣ

Στη συνέχεια θα περιγράψουμε πως μπορεί ο δενδρικός αλγόριθμος Παπαρρίζου με ξεκίνημα το δέντρο Balinski να εφαρμοστεί για τη επίλυση του προβλήματος Αντιστοίχησης **[P1]**. Θα παραθέσουμε την περιγραφή του αλγορίθμου , τη βηματική του εκτέλεση σε μορφή ψευδοκώδικα καθώς και λεπτομέρειες για έναν αποτελεσματικό τρόπο προγραμματισμού του αλγορίθμου.

3.2.1 Περιγραφή του δέντρου Balinski για το πρόβλημα Αντιστοίχησης

Όπως αναφέρθηκε και στην εισαγωγή , το πρόβλημα Αντιστοίχησης είναι μια ειδική περίπτωση του προβλήματος Μεταφοράς από το οποίο και προκύπτει θέτοντας $m = n$ και $a_i = b_i = 1,\ i = 1..n$.[3] Ο δενδρικός αλγόριθμος Παπαρρίζου για το πρόβλημα Αντιστοίχησης με ξεκίνημα το δέντρο Balinski αποτελεί στην ουσία εξειδίκευση του αντίστοιχου αλγορίθμου για το πρόβλημα Μεταφοράς. Στη συνέχεια θα περιγραφεί το δέντρο Balinski με τρόπο εξειδικευμένο για το πρόβλημα Αντιστοίχησης.

Ο τρόπος υπολογισμού του δέντρου Balinski είναι ακριβώς ο ίδιος με αυτόν στο πρόβλημα Μεταφοράς. Η μόνη διαφορά έγκειται στον τρόπο υπολογισμού του συνόλου F. Όπως είχαμε δει στο πρόβλημα Μεταφοράς, το σύνολο F περιείχε τους κόμβους των υποδέντρων T_j, ώστε $x_{1j} < 0$. Αυτό φαίνεται αναλυτικά στο παρακάτω σχήμα:



Εικόνα 3.8 – Προσδιορισμός του x_{1j} στο πρόβλημα Μεταφοράς.

Στο παραπάνω σχήμα είναι $x_{1j} < 0$ αν και μόνον αν $b_j < \sum_{(i,j)\in T} x_{ij}$. Δηλαδή η τιμή x_{1j} είναι μικρότερη του μηδενός μόνο όταν το άθροισμα όλων των ροών που φτάνουν στον κόμβο j είναι μεγαλύτερο από τη ζήτηση του συγκεκριμένου κόμβου. Αυτή η παρατήρηση σε συνδυασμό με το γεγονός ότι στο πρόβλημα Αντιστοίχησης ισχύει $a_i = b_i = 1,\ i = 1,...,n$ εύκολα μας οδηγεί στο συμπέρασμα ότι τα δέντρα T_j στο πρόβλημα Αντιστοίχησης θα ανήκουν στο σύνολο F (δηλαδή το j θα είναι τέτοιο ώστε $x_{1j} < 0$) αν και μόνον αν ο βαθμός του κόμβου j είναι μεγαλύτερος του 3. Σε αυτήν την περίπτωση, θα έχουμε $k \geq 2$ τόξα της μορφής $(s, j),\ s \neq 1$ και το μοναδικό προς τα κάτω τόξο $(1, j)$. Έτσι, για $k \geq 2$ ο βαθμός του κόμβου j θα είναι μεγαλύτερος του $3 (d(j) \geq 3)$. Συνεπώς, είμαστε τώρα σε θέση να ορίσουμε το σύνολο F και συνεπώς θα έχουμε:

$$F = \left\{ T_j : d(j) \geq 3 \right\}$$

[3] Δηλαδή, η μήτρα συντελεστών ενός προβλήματος Αντιστοίχησης είναι τετραγωνική

Οι δυϊκές μεταβλητές προσδιορίζονται ακριβώς με τον ίδιο τρόπο όπως και στο πρόβλημα Μεταφοράς. Στη συνέχεια θα λύσουμε ένα παράδειγμα.

Παράδειγμα 3.3

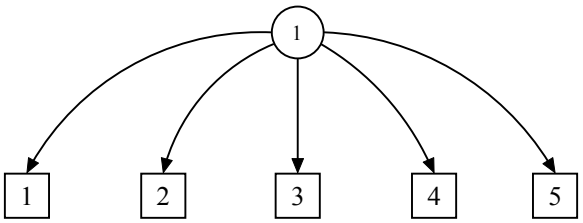
Δίνεται ο πίνακας κόστους

$$C=\begin{bmatrix} 3 & -3 & 19 & 10 & -27 \\ 0 & -1 & 42 & 46 & 77 \\ 70 & 32 & 43 & 71 & 90 \\ 34 & 1 & 0 & 0 & 8 \\ 1 & 7 & 34 & 65 & -7 \end{bmatrix}$$

για ένα 5×5 ***πρόβλημα Αντιστοίχησης. Να προσδιορισθεί το δέντρο ξεκινήματος Balinski και να υπολογισθούν τα στοιχεία*** $x_{ij}(T),\ (i,j)\in T$**,** $u_i(T),\ i=1..n$**,** $v_j(T),\ j=1,...,n$ ***και*** $s_{ij}(T),\ (i,j)\notin T$.

Λύση

Έστω T το δέντρο Balinski. Αρχικά θέτουμε $u_1(T)=0$ και $v_j(T)=c_{1j},\ j=1,...,5$. Έτσι οι τιμές των μεταβλητών που προκύπτουν σύμφωνα με τον πίνακα των δεδομένων θα είναι: $u_1(T)=0,\ v_1(T)=3,\ v_2(T)=-3,\ v_3(T)=19,\ v_4(T)=10,\ v_5(T)=-27$. Συνεπώς, μέχρι τώρα το δέντρο που έχει σχηματισθεί θα είναι το παρακάτω:



Εικόνα 3.9 – Αρχικό στάδιο κατασκευής του δέντρου Balinski

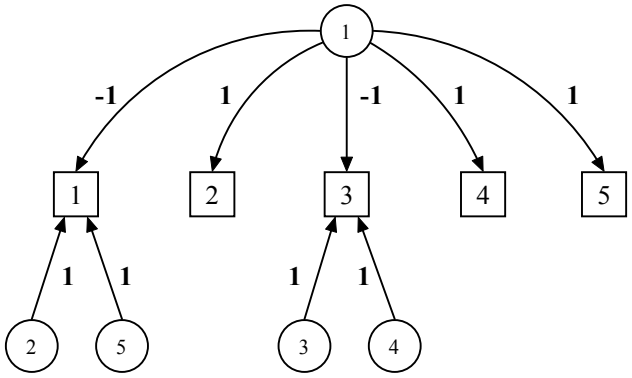
Για όλα τα τόξα (i,j) του παραπάνω δέντρου θα έχουμε $s_{ij}(T)=c_{ij}-u_i(T)-v_j(T)=0$. Για τον υπολογισμό των άλλων βασικών τόξων του δέντρου αφαιρούμε την τιμή της δυϊκής μεταβλητής v_j από όλα τα στοιχεία της στήλης j του πίνακα κόστους. Έτσι προκύπτει ο πίνακας

$$D=\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ -3 & 2 & 23 & 36 & 104 \\ 67 & 35 & 24 & 61 & 117 \\ 31 & 4 & -19 & -10 & 35 \\ -2 & 10 & 15 & 55 & 20 \end{bmatrix}$$

Συνεπώς τα νέα βασικά τόξα $(i,j(i))$ του δέντρου Balinski θα είναι αυτά που ικανοποιούν τη σχέση

$$c_{ij(i)}-v_{j(i)}(T)=\min\left\{D^i{}_j : j=1,...,n\right\}\ ,\ i=2,...,m$$

, όπου D^i η i γραμμή του πίνακα D. Με αυτόν τον τρόπο διαπιστώνουμε ότι τα νέα βασικά τόξα του δέντρου Balinski θα είναι τα τόξα (3,3), (4,3), (2,1) και (5,1). Επίσης, οι δυϊκές μεταβλητές των υπολοίπων κόμβων γραμμών θα είναι $u_2 = -3$, $u_3 = 24$, $u_4 = -19$ και $u_5 = -2$. Συνεπώς τα διανύσματα των δυϊκών μεταβλητών θα είναι $u = [0 \quad -3 \quad 24 \quad -19 \quad -2]$ και $v = [3 \quad -3 \quad 19 \quad 10 \quad -27]$. Θέτουμε επίσης $x_{33} = x_{43} = x_{21} = x_{51} = 1$. Στη συνέχεια θέτουμε $x_{11} = x_{13} = -1$ αφού $d(1) = d(3) = 3 \geq 3$ και $x_{12} = x_{14} = x_{15} = 1$. Εύκολα τώρα παράγουμε τον πίνακα των μειωμένων κοστών ο οποίος περιέχει και τις μεταβλητές απόφασης στα σκιασμένα κελιά. Το δέντρο Balinski και ο αντίστοιχος πίνακας φαίνονται στο παρακάτω σχήμα:



-1	**1**	**-1**	**1**	**1**
1	5	26	39	107
43	11	**1**	37	93
50	23	**1**	9	54
1	12	17	57	22

Εικόνα 3.10 – Το δέντρο Balinski και ο αρχικός πίνακας του προβλήματος.

Το σύνολο F θα αποτελείται από τα δέντρα T_1, T_3, αφού $d(1) = d(3) = 3$.

3.2.2 Περιγραφή του Αλγορίθμου

Ο δενδρικός αλγόριθμος Παπαρρίζου για το πρόβλημα Αντιστοίχησης ξεκινά με το δέντρο Balinski. Είναι στην ουσία ο ίδιος αλγόριθμος με αυτόν που χρησιμοποιείται στο πρόβλημα Μεταφοράς με τη διαφορά ότι γίνονται ορισμένες εξειδικεύσεις για την επίτευξη καλύτερων υπολογιστικών αποδόσεων. Έστω T λοιπόν το τρέχον δέντρο του αλγορίθμου. Αν ισχύει $x_{1j}(T) \geq 0$ για κάθε βασικό τόξο $(1, j)$ του δέντρου T, ο αλγόριθμος σταματάει και T είναι το βέλτιστο δέντρο. Διαφορετικά προσδιορίζονται τα τόξα $(1, j)$ τέτοια ώστε $x_{1j}(T) < 0$. Σχηματίζουμε λοιπόν το σύνολο J ως εξής:

$$J = \{ j : \ j \in D \wedge d(j) \geq 3 \}$$

, όπου D το σύνολο των κόμβων ζήτησης του προβλήματος. Έστω τώρα T_j το υπόδεντρο του T που κόβεται από τη ρίζα, όταν διαγράφεται το τόξο $(1, j) \in T$. Συμβολίζουμε με F το

δάσος που αποτελείται από δέντρα T_j τέτοια ώστε $j \in J$. Το εισερχόμενο τόξο (g,h) προσδιορίζεται από την παρακάτω σχέση:

$$\delta = s_{gh} = \min\left\{s_{ij}(T) : i \in F, j \in T \sim F\right\} \geq 0$$

Παρατηρούμε ότι για τον προσδιορισμό του συνόλου F ελέγχουμε το βαθμό των κόμβων στηλών του δέντρου. Θυμίζουμε ότι ο βαθμός $d(x)$ ενός κόμβου x ισούται με το πλήθος των τόξων που εισέρχονται και εξέρχονται από αυτόν. Θα μπορούσαμε χωρίς καμιά διαφορά να ελέγχουμε την αντίστοιχη τιμή $x_{1j}(T)$ και ανάλογα με το πρόσημό της να εισάγουμε ή όχι τον αντίστοιχο κόμβο στο σύνολο F. Αυτό το κάνουμε γιατί όπως θα δούμε ο βαθμός των κόμβων θα χρειαστεί ούτως ή άλλως να προσδιορισθεί και παρακάτω έτσι ώστε να επιλεχθεί το αντίστοιχο εξερχόμενο τόξο (k,ℓ).

Για τον προσδιορισμό του εξερχόμενου τόξου (k,ℓ) διακρίνουμε τις παρακάτω δύο περιπτώσεις. Έστω C ο μοναδικός κύκλος του γραφήματος $T \cup (g,h)$. Έστω επίσης $f \in F$ ο κόμβος στήλη του κύκλου C που κρέμεται από τη ρίζα του δέντρου Balinski. Συνεπώς το τόξο $(1,f)$ περιέχεται στο δρόμο από τον κόμβο g προς τη ρίζα του δέντρου $P[T,g]$. Το τόξο $(1,f)$ θα περιέχεται συνεπώς στον κύκλο C. Αν $d(h)=2$, τότε εξερχόμενο είναι το μοναδικό προς τα κάτω τόξο (k,h), αλλιώς αν $d(h)=1 \wedge d(f)>3$, τότε εξερχόμενο τόξο είναι το μοναδικό προς τα πάνω τόξο (i,f), αλλιώς αν $d(h)=1 \wedge d(f)=3$, το εξερχόμενο τόξο είναι το μοναδικό τόξο $(1,f) \in C$.

Τέλος, ως γνωστόν ανανεώνονται τα μειωμένα κόστη τόξων που προσπίπτουν σε κόμβους που ανήκουν στο αποκομμένο δέντρο $T*$. Η πλήρης βηματική περιγραφή του αλγορίθμου φαίνεται παρακάτω.

3.2.3 Βηματική Περιγραφή του Αλγορίθμου

Στη συνέχεια θα παρουσιάσουμε τον αλγόριθμο σε μορφή βημάτων.

ΒΗΜΑ 0:*(Καθορισμός αρχικών μεταβλητών)*
Υπολόγισε το δέντρο Balinski T με τη μέθοδο που περιγράψαμε και καθόρισε το σύνολο F τέτοιο $i \in F$ αν και μόνον αν $i \in T_j$ τέτοιο ώστε $d(j) \geq 3$.(ή αντίστοιχα $x_{1j} < 0$)

ΒΗΜΑ 1:*(Έλεγχος βελτιστότητας)*

βέλτιστο και ο αλγόριθμος τερματίζει, αλλιώς πήγαινε στο βήμα 2.

ΒΗΜΑ 2:*(Επιλογή εισερχομένου τόξου)*
Υπολόγισε το εισερχόμενο τόξο (,) ελέγχοντας τα μειωμένα κόστη των μη βασικών
F

ΒΗΜΑ 3:*(Επιλογή εξερχόμενου τόξου)*

όπου (i,f) το μοναδικό προς τα πάνω τόξο που ανήκει στον κύκλο C, αλλιώς αν $d(h)=1 \wedge d(f)=3$ θέσε $(k,\ell)=(1,f$.
ΒΗΜΑ 4:*(Ανανέωση των μεταβλητών)*

Υπολόγισε το δέντρο $T*$ που είναι το δέντρο που αποκόβεται από τη ρίζα όταν αφαιρείται το εξερχόμενο τόξο. Αν $h \in T*$ θέσε $q = -\delta > 0$, αλλιώς θέσε $q = \delta < 0$. Στη συνέχεια εάν ο κόμβος $b \in T*$ είναι κόμβος-γραμμή θέσε $s_{bj}(T') = s_{bj}(T) - q$, $j = 1..n$ αλλιώς (δηλαδή ο κόμβος $b \in T*$ είναι κόμβος στήλη θέσε $s_{ib}(T') = s_{ib}(T) + q$, $i = 1..n$.

ΒΗΜΑ 5:*(Ανανέωση του τρέχοντος δέντρου)*

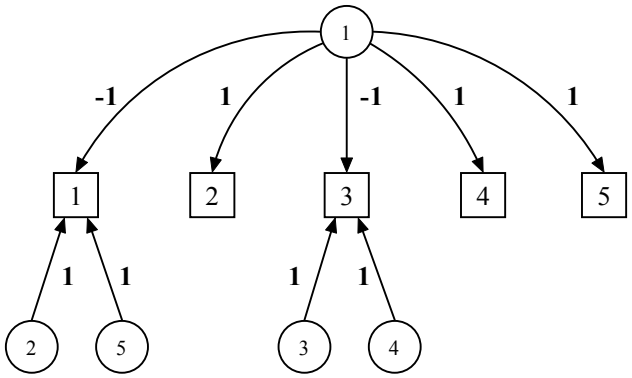
Παράδειγμα 3.4
Δίνεται ο πίνακας κόστους

$$C = \begin{bmatrix} 3 & -3 & 19 & 10 & -27 \\ 0 & -1 & 42 & 46 & 77 \\ 70 & 32 & 43 & 71 & 90 \\ 34 & 1 & 0 & 0 & 8 \\ 1 & 7 & 34 & 65 & -7 \end{bmatrix}$$

για ένα 5×5 ***πρόβλημα Αντιστοίχησης. Να λυθεί το πρόβλημα χρησιμοποιώντας το δεντρικό αλγόριθμο Παπαρρίζου με ξεκίνημα το δέντρο Balinski.***

Λύση

Αρχικά θα πρέπει να υπολογίσουμε το εφικτό δέντρο Balinski και τον πίνακα των μειωμένων κοστών του προβλήματος. Όλα αυτά έγιναν στο παράδειγμα 3.3 και φαίνονται παρακάτω:

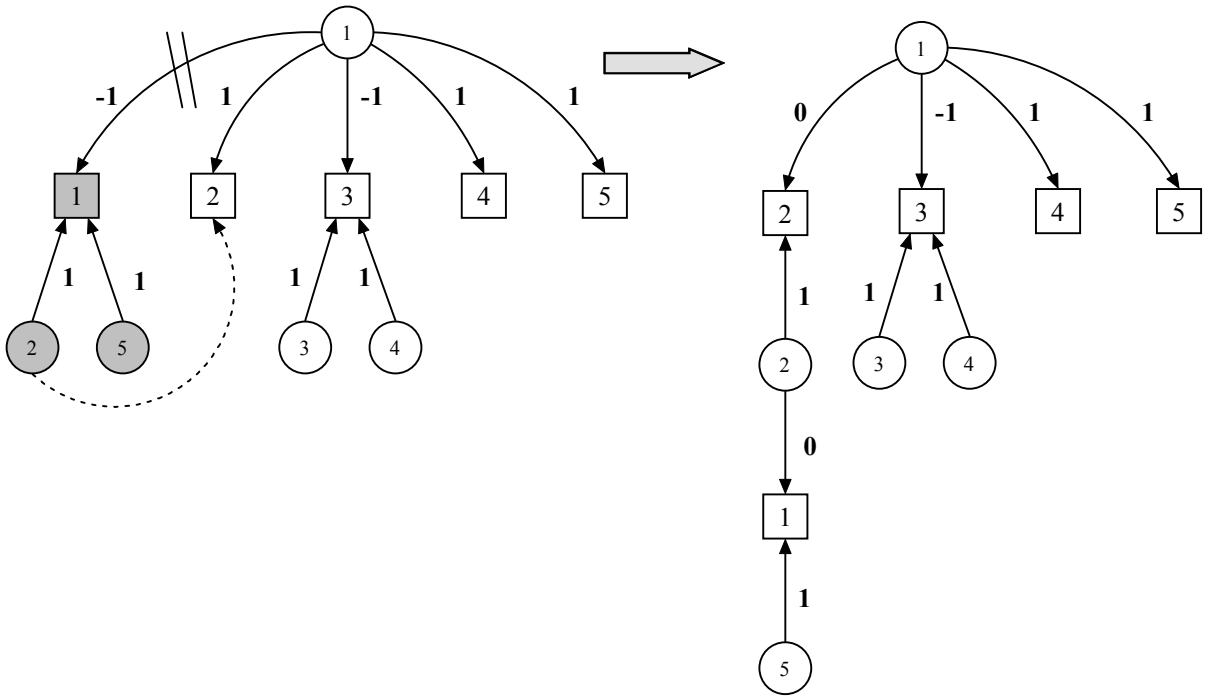


-1	1	-1	1	1
1	5	26	39	107
43	11	**1**	37	93
50	23	**1**	9	54
1	12	17	57	22

Εικόνα 3.11 – Το εφικτό δέντρο Balinski και η αντίστοιχη μήτρα μειωμένων κοστών.

Επανάληψη 1

- Είναι $F=\{2,3,4,5\}$, $T\sim F=\{2,4,5\}$, $\delta=\min\{s_{ij}(T): i\in F, j\in T\sim F\}=s_{22}=5$, συνεπώς $(g,h)=(2,2)$ [4].
- Σύμφωνα με όσα αναφέρθηκαν στην περιγραφή του αλγορίθμου θα πρέπει να βρούμε το βαθμό του κόμβου h. Είναι $d(h)=d(2)=1$, άρα θα πρέπει να ελέγξουμε και το βαθμό του κόμβου f, όπου f είναι ο μοναδικός κόμβος - στήλη που κρέμεται από τη ρίζα του δέντρου και ανήκει στον κύκλο C της επανάληψης. Συνεπώς $f=1$ και $d(f)=3$, άρα εξερχόμενο θα είναι το τόξο $(k,\ell)=(1,f)=(1,1)$.
- Η ανανέωση των τιμών των μεταβλητών απόφασης και των μειωμένων κοστών γίνεται με βάση τον κύκλο C και του αποκομμένου δέντρου $T*$ και φαίνεται αναλυτικά στο σχήμα 3.12.

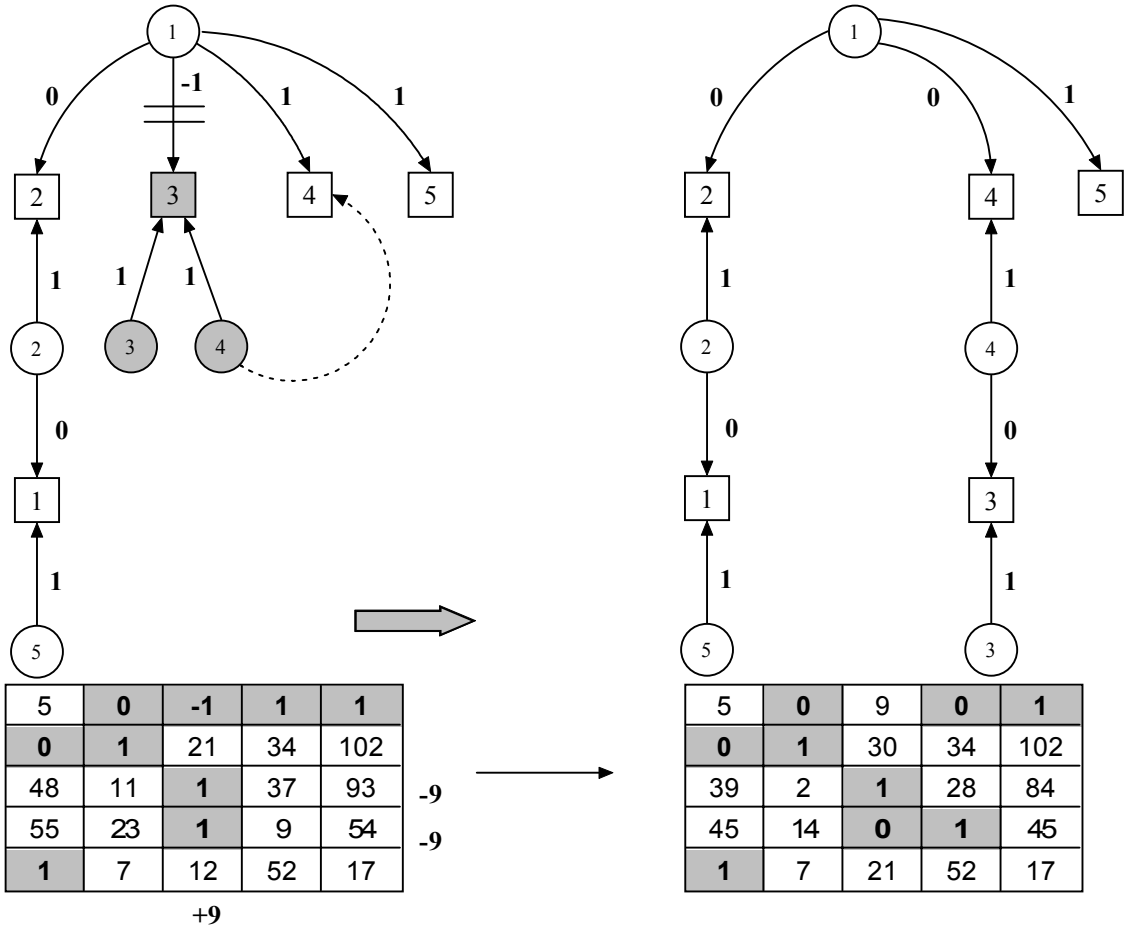


Εικόνα 3.12 – Η πρώτη επανάληψη του αλγορίθμου

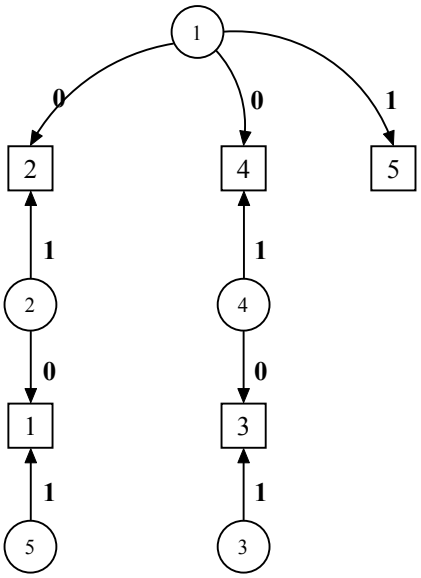
Επανάληψη 2

[4] Παρατηρείστε ότι στο σύνολο F αποθηκεύουμε μόνο τους κόμβους γραμμές που ανήκουν σε αυτό, επειδή από το σύνολο F μπορεί να προέλθει μόνο η άκρη g του εισερχομένου τόξου (g,h) που είναι πάντα κόμβος γραμμή. Το ίδιο γίνεται και με το σύνολο $T\sim F$ στο οποίο για τον ίδιο ακριβώς λόγο αποθηκεύουμε μόνο τους κόμβους στήλες. Αυτή η τεχνική θα χρησιμοποιηθεί και στους επόμενους αλγορίθμους που θα παρουσιαστούν. Ωστόσο, η ολοκληρωμένη και μαθηματικά ορθή αναπαράσταση των προαναφερθέντων συνόλων θα ήταν η $F=\{2,3,4,5,6,8\}$, $T\sim F=\{7,9,10\}$, όπου ο κόμβος-στήλη x αποθηκεύεται ως $x+n$.

- Είναι $F=\{3,4\}$, $T\sim F=\{1,2,4,5\}$, $\delta=\min\{s_{ij}(T): i\in F, j\in T\sim F\}=s_{44}=9$, συνεπώς $(g,h)=(4,4)$.
- Είναι $d(h)=d(2)=1$, άρα θα πρέπει να ελέγξουμε και το βαθμό του κόμβου f. Είναι $d(f)=d(3)=3$, άρα $(k,\ell)=(1,f)=(1,3)$.
- Το δέντρο $T*$ θα αποτελείται από τους κόμβους γραμμές 3, 4 και τον κόμβο στήλη 3



Εικόνα 3.13 – Η δεύτερη επανάληψη του αλγορίθμου



Εικόνα 3.14 – Το βέλτιστο δέντρο του αλγορίθμου

Η ανανέωση των μεταβλητών του αλγορίθμου της δεύτερης επανάληψης φαίνονται στο σχήμα 3.13.Μετά το πέρας και της δεύτερης επανάληψης, το δέντρο που παράγεται είναι βέλτιστο, αφού δεν υπάρχει κόμβος στήλη j τέτοιος ώστε $d(j) \geq 3$. Συνεπώς $F = \varnothing$ και ο αλγόριθμος τερματίζει αφού ικανοποιείται ο έλεγχος βελτιστότητας. Το βέλτιστο δέντρο που παράγεται φαίνεται ξεχωριστά στο σχήμα 3.14. Το βέλτιστο διάνυσμα ανάθεσης (βέλτιστη μετάθεση) είναι το $p = [5 \quad 2 \quad 3 \quad 4 \quad 1]$ ενώ η βέλτιστη τιμή της αντικειμενικής συνάρτησης θα είναι $z = \sum_{(i,j)\in T} c_{ij} x_{ij} = 16$.

3.2.4 Προγραμματισμός του Αλγορίθμου

Στη συνέχεια θα παρουσιάσουμε έναν αποτελεσματικό τρόπο προγραμματισμού του αλγορίθμου. Θα παρουσιάσουμε αρχικά το «κυρίως πρόγραμμα» του αλγορίθμου και στη συνέχεια θα αναλύσουμε τις σημαντικότερες συναρτήσεις που χρησιμοποιήσαμε για την υλοποίησή του.

Αλγόριθμος Α2
Ο παρακάτω αλγόριθμος λύνει το πρόβλημα Αντιστοίχησης , χρησιμοποιώντας το δενδρικό αλγόριθμο Παπαρρίζου με ξεκίνημα το δέντρο Balinski.
Είσοδος: *Ο πίνακας κόστους* **C(nxn)**.
Έξοδος : *Η λύση του προβλήματος Αντιστοίχησης* **X(nxn)**.

```
1.  [X,S]=BalinskiTree(C,n);
2.  p=InitializeNodeFather(X,n);
3.  d=InitializeNodeDepth(p,n);
4.  deg=InitializeDegree(X,n);
5.  stopalgorithm=0;
6.  while stopalgorithm~=1
7.    [F,Fbar]=ForestsF(X,p,n);
8.    [min,g,h,f,Circle]=EnteringArc(S,F,Fbar,n,p);
9.    [k,l,p,upwards,Tcut,deg]=LeavingArc(g,h,deg,f,p,n,Circle);
10.   if (length(Tcut)~=0)
11.     [e1,e2,f1,z]=Steam(p,g,h,k,l,d,n,Tcut);
12.     S=UpdateReducedCosts(min,h,Tcut,n,S);
13.     d=UpdateNodeDepth(p,n,Tcut,z,d,e1,e2,f1);
14.     p=UpdateNodeFather(p,z,e2,f1);
15.     X=UpdateFlows(X,d,k,l,g,h,upwards,n);
16.   end
17.   fw=find(X(1,:)>=0);
18.   if length(fw)==n
19.      stopalgorithm=1;
20.   end
21. end
```

Αλγόριθμος 3.2.1 – Ο κώδικας του δενδρικού αλγόριθμου Παπαρρίζου με ξεκίνημα το δέντρο Balinski για το πρόβλημα Αντιστοίχισης

Στον παραπάνω κώδικα μπορούμε να παρατηρήσουμε τα εξής: Στις πρώτες 4 γραμμές, υπολογίζεται το δέντρο Balinski και έτσι επιστρέφονται οι αρχικές μήτρες των μεταβλητών απόφασης και των μειωμένων κοστών. Επίσης αρχικοποιούνται τα διανύσματα πατέρα-κόμβου, βάθους και βαθμών των κόμβων. Το διάνυσμα βαθμών **`deg()`** που επιστρέφει η συνάρτηση **`InitializeDegree()`** είναι ένα διάνυσμα n θέσεων που περιέχει τους βαθμούς μόνο των κόμβων- στηλών που ανήκουν στο τρέχον δέντρο, αφού για την επιλογή του εξερχόμενου τόξου (k,ℓ), ελέγχουμε τον βαθμό μόνο του κόμβου στήλη h. Στις επόμενες γραμμές του κώδικα αρχίζει η εκτέλεση του αλγορίθμου. Αρχικά, υπολογίζονται τα σύνολα F και $T \sim F$ μέσω της συνάρτησης **`ForestsF()`** (γραμμή 7). Κατόπιν επιλέγεται το εισερχόμενο και εξερχόμενο τόξο μέσω των συναρτήσεων **`EnteringArc()`** και **`LeavingArc()`** και γίνονται οι απαραίτητες ανανεώσεις των μεταβλητών. Τέλος, στις

γραμμές 17-25 του κώδικά μας γίνεται ο έλεγχος τερματισμού. Έτσι ελέγχεται το σύνολο F και αναλόγως ο αλγόριθμος συνεχίζει ή τερματίζει. Προσέξτε ότι ο έλεγχος του συνόλου F γίνεται έμμεσα μέσω του πρόσημου των τιμών x_{1j}. Έτσι, αν $x_{1j} \geq 0 \quad \forall (1,j) \in T$ (ο έλεγχος θα μπορούσε να γίνει χρησιμοποιώντας και την αντίστοιχη συνθήκη $d(j) < 3 \quad \forall (1,j) \in T$), τότε το τρέχον δέντρο T είναι βέλτιστο και ο αλγόριθμος τερματίζει (γραμμή 19), αλλιώς ο αλγόριθμος συνεχίζει την εκτέλεσή του. Στη συνέχεια θα αναλυθούν οι σημαντικότερες συναρτήσεις που χρησιμοποιήσαμε για την υλοποίηση του αλγορίθμου:

- **[k, l, p, upwards, Tcut, deg] = LeavingArc (g, h, deg, f, p, n, Circle)**

Αλγόριθμος LeavingArc
Ο αλγόριθμος που υπολογίζει μεταβλητές που σχετίζονται με το εξερχόμενο τόξο.
Είσοδος: *Τα άκρα του εισερχόμενου τόξου* **g,h** *, το διάνυσμα βαθών* **deg(n)**, **f** ,**p(2*n), n**, **Circle()**
Έξοδος : *Τα άκρα του εξερχόμενου τόξου* **k, l** *το ανανεωμένο* **p(2*n)** *, μια μεταβλητή που δείχνει τη φορά του εξερχόμενου τόξου* **upwards***, το διάνυσμα του αποκομμένου δέντρου* **Tcut()***, το διάνυσμα των κόμβων που ανήκουν στον κύκλο* **Circle()**.

```
1.  function [k,l,p,upwards,Tcut,deg]=LeavingArc(g,h,deg,f,p,n,Circle)
2.  if deg(h)==2
3.    k=p(h+n);
4.    l=h;
5.    start=l+n;
6.    Tcut=Preorder(start,p,n);
7.    upwards=0;
8.  end
9.  if deg(h)==1
10.   if deg(f-n)>3
11.     upwards=1;
12.     i=1;
13.     while Circle(i)~=0
14.       if p(Circle(i))==f
15.         k=Circle(i);
16.         l=f-n;
17.       end
18.       i=i+1;
19.     end
20.     start=k;
21.     Tcut=Preorder(start,p,n);
22.  elseif deg(f-n)==3
23.     k=1;
24.     l=f-n;
25.     start=f;
26.     Tcut=Preorder(start,p,n);
27.     upwards=1;
28.  end
29. deg(f-n)=deg(f-n)-1;
30. deg(h)=deg(h)+1;
31. end
```

Αλγόριθμος 3.2.2 – Επιλογή του εξερχόμενου τόξου με βάση τους βαθμούς των κόμβων - στηλών.

Στον παραπάνω αλγόριθμο γίνεται η επιλογή του εξερχόμενου τόξου με βάση το βαθμό του κόμβου-στήλη h, όπως ακριβώς είδαμε στην περιγραφή του αλγορίθμου. Ο βαθμός του κόμβου-στήλη h αποθηκεύεται στη θέση h του διανύσματος **deg()**. Στις γραμμές 2-8 του κώδικα εξετάζεται η περίπτωση όπου $d(h) = 2$. Σε αυτήν την περίπτωση θέτουμε $k = p(h+n)$ και $\ell = h$, όπως ακριβώς είδαμε και στην περιγραφή του αλγορίθμου. Στη συνέχεια (γραμμές 9-28), παρατηρούμε ότι εξετάζεται η περίπτωση όπου $d(h) = 1$.

Ειδικότερα, βλέπουμε ότι ο αλγόριθμος σε αυτήν την περίπτωση υπολογίζει τον κόμβο f και αναλόγως με το βαθμό του κόμβου f υπολογίζει το εξερχόμενο τόξο (k,ℓ).

Αυτό που πρέπει να τονισθεί για τον παραπάνω κώδικα είναι ο τρόπος ανανέωσης του διανύσματος των βαθμών των κόμβων του δέντρου. Σε κάθε επανάληψη του αλγορίθμου, μπορεί να αλλάζουν τιμή μόνο οι βαθμοί των κόμβων h, f. Το διάνυσμα των βαθμών του δέντρου ανανεώνεται ανάλογα με το βαθμό του κόμβου h. Συγκεκριμένα, αν $d(h)=2$, τότε δεν επέρχεται καμία μεταβολή στο βαθμό του κόμβου h, πράγμα πολύ λογικό αν σκεφτεί κανείς ότι σε αυτήν την περίπτωση και το εισερχόμενο τόξο και το εξερχόμενο προσπίπτουν στον κόμβο h. Στην αντίθετη περίπτωση, δηλαδή όταν $d(h)=1$, τότε ο βαθμός του κόμβου f μειώνεται κατά 1, ενώ ο βαθμός του κόμβου h αυξάνεται κατά 1 (γραμμές 29-30). Αυτό φαίνεται άλλωστε καλύτερα και στις επαναλήψεις του παραδείγματος(εικόνες 3.12, 3.13).

- **deg = InitializeDegree(X , n)**

Αλγόριθμος InitializeDegree
Ο αλγόριθμος που αρχικοποιεί το διάνυσμα των βαθμών των κόμβων.
Είσοδος: *Η μήτρα των μεταβλητών απόφασης* **X(nxn)**, *η διάσταση του προβλήματος* **n**.
Έξοδος : *Το αρχικοποιημένο διάνυσμα των βαθμών των κόμβων* **deg(n)**

```
1.  function deg=InitializeDegree(X,n)
2.  for j=1:n
3.    deg(j)=0;
4.  end
5.  for j=1:n
6.    s=0;
7.    for i=1:n
8.      if X(i,j)~=inf
9.        s=s+1;
10.       deg(j)=s;
11.     end
12.   end
13. end
```

Αλγόριθμος 3.2.3 – Αρχικοποίηση του διανύσματος βαθμών των κόμβων.

Η παραπάνω συνάρτηση χρησιμοποιήθηκε για την αρχικοποίηση των βαθμών των κόμβων του δέντρου. Παρατηρούμε ότι χρησιμοποιεί την μήτρα των μεταβλητών απόφασης, ελέγχοντας που υπάρχει τόξο και αναλόγως αυξάνει το βαθμό του αντίστοιχου κόμβου. Η κλήση αυτής της διαδικασίας σε κάθε επανάληψη για τον υπολογισμό των βαθμών των κόμβων αποφεύγεται λόγω της αποτελεσματικής μεθόδου ανανέωσης των βαθμών των κόμβων που περιγράφηκε στον αλγόριθμο 3.2.2.

- **d = InitializeNodeDepth(p , n)**

Αλγόριθμος InitializeNodeDepth
Ο αλγόριθμος που αρχικοποιεί το διάνυσμα των βαθών των κόμβων.
Είσοδος:*Το διάνυσμα πατέρα-κόμβου* **p(2*n)** , *η διάσταση του προβλήματος* **n**.
Έξοδος : *Το αρχικοποιημένο διάνυσμα βάθους των κόμβων* **d(n)**

```
1.  function d=InitializeNodeDepth(p,n)
2.  d(1)=0;f=0;
3.  while (f~=2*n)
4.    f=0;
5.    for i=2:2*n
6.      d(i)=d(p(i))+1;
```

```
7.    end
8.    for u=1:2*n
9.      if d(u)~=inf
10.       f=f+1;
11.     end
12.   end
13. end
```

Αλγόριθμος 3.2.4 – Ο αλγόριθμος αρχικοποίησης των βαθών των κόμβων

Στον αλγόριθμο **InitializeNodeDepth** παρουσιάζεται ένας τρόπος αρχικοποίησης των βαθών των κόμβων του δέντρου. Αρχικά τίθεται **d(1)=0** (γραμμή 2), αφού το βάθος της ρίζας ενός δέντρου είναι πάντα 0 και ρίζα του δέντρου Balinski είναι ο κόμβος-γραμμή 1. Στη συνέχεια, ο αλγόριθμος θέτει το βάθος κάθε κόμβου ίσο με το βάθος του πατέρα του αυξημένο κατά 1 (γραμμή 6). Τέλος ο αλγόριθμος τερματίζει όταν υπολογιστούν τα βάθη όλων των κόμβων. Η ανανέωση των βαθών γίνεται με τον ίδιο αποτελεσματικό τρόπο που περιγράφηκε και στον αλγόριθμο Simplex.

3.3 ΑΝΑΦΟΡΕΣ

[P1] K. Paparrizos, *"An Infeasible (Exterior Point) Simplex Algorithm for Assignment Problems"*, Mathematical Programming 51 (1991) 45-54 North Holland

[P2] K. Paparrizos, *"A non improving simplex algorithm for transportation problems"*, Operations Research, vol 30, pp 1-15,1996

[P3] H. Achatz, P. Kleinschmidt and K. Paparrizos, "*A dual forest algorithm for the assignment problem,*" Dimacs, vol. 4, pp. 1-10, 1990.

[B1] M.L. Balinski,"*The Hirsch conjecture for dual transportation polyhedra*", Math. Oper. Res. 9 (1984), 629-633

[B2] M.L. Balinski,"*Signature Methods for the assignment problem",* Oper. Res. 33 (1985), 527-537

[B3] M.L. Balinski,"*A competitive (dual) simplex method for the assignment problem",* Math. Programming 34 (1986), 125-141

[B4] M. L. Balinski and R. E. Gomory, *"A primal method for the assignment and transportation problems"* , Management Sciences, vol. 10, pp. 578-598, 1964.

[Cun] Cunningham, W.H. (1976),"*A Network Simplex Method"*, Math Prog 11,105-116

[JB] Jensen and Barnes, (1980),*"Network Flow Programming"*, Malabar, Fla.: R.E. Krieger Pub. Co. , 1980

[Ber] Bertsekas D.**, "***Network Optimization: Continuous and Discrete Models"*, Athena Scientific, 1998

[Roc] Rockafellar R.T., **"***Network Flows and Monotropic Optimization"* , Wiley-Interscience, 1984 (610 pp.).

[CLR] Cormen , Leiserson , Rivest, *"Introduction to Algorithms", MIT Press*, (1990)

[Kn] D.E. Knuth, *"The Art of Computer Programming – Fundamental Algorithms"*, 3rd edition, Addison-Wesley, 1997

[Pap1] K. Paparrizos, *"Linear Programming – Algorithms and Applications"*, in greek

[CP] Papadimitriou C., *"Combinatorial Optimization – Algorithms and Complexity"*, Prentice Hall (1982)

[Pap2] K. Paparrizos, *"Network Programming"*, lecture notes in Greek

[Pap3] K. Paparrizos, *"Algorithms – Analysis, Design and Complexity"*, lecture notes in Greek

[SS] Stephanides G., Samaras N., *"Computational Methods with Matlab"*, Zygos Publications, Thessaloniki 1999, in Greek

[PK] Paparrizos K., *"Matlab 6"*, Zygos Publications, Thessaloniki 2001, in Greek

ΚΕΦΑΛΑΙΟ 4

Ο ΑΛΓΟΡΙΘΜΟΣ ΠΑΠΑΡΡΙΖΟΥ

(ΞΕΚΙΝΗΜΑ ΜΕ ΔΑΣΟΣ ΑΚΡ)

Στο κεφάλαιο αυτό θα παρουσιάσουμε τον αλγόριθμο Παπαρρίζου με ξεκίνημα το δάσος AKP (*"Achatz-Kleinsschmidt-Paparrizos"*). Η φιλοσοφία αυτού του αλγορίθμου είναι ίδια με τη αυτήν που παρουσιάσθηκε στο προηγούμενο κεφάλαιο. Είναι και αυτός ένας αλγόριθμος εξωτερικών σημείων(*exterior point algorithm*). Στο δρόμο του δηλαδή προς το βέλτιστο σημείο, περνάει από σημεία που δεν ανήκουν στο πολύεδρο της εφικτής περιοχής. Η διαφορά αυτού του αλγορίθμου από τον αλγόριθμο του προηγούμενου κεφαλαίου έγκειται στην κατασκευή του δάσους ξεκινήματος. Έτσι, ο αλγόριθμος που θα περιγραφεί χρησιμοποιεί ένα δάσος ξεκινήματος που είναι πάρα πολύ αποτελεσματικό από υπολογιστικής απόψεως. Συγκεκριμένα, σε πρόσφατη υπολογιστική μελέτη **[PPS]**, ο αλγόριθμος Παπαρρίζου με ξεκίνημα το δάσος AKP εμφανίζεται μέχρι και 3 φορές καλύτερος σε χρόνο CPU από τον πρωτεύοντα αλγόριθμο Simplex για δίκτυα, ενώ υπερτερεί και των άλλων αλγορίθμων εξωτερικών σημείων. Η σύγκριση των αλγορίθμων αυτών θα παρουσιασθεί διεξοδικά στο Κεφάλαιο 6. Ο αλγόριθμος με το συγκεκριμένο δάσος ξεκινήματος αναπτύχθηκε για το πρόβλημα αντιστοίχησης από τον καθηγητή του τμήματος Κ. Παπαρρίζο σε συνεργασία με τους συναδέλφους του H.Achatz και P.Kleinsschmidt του τμήματος Μαθηματικών και Πληροφορικής του Πανεπιστημίου Passau, παρουσιάζοντας συγχρόνως θεωρητικά αποτελέσματα και μια προκαταρτική υπολογιστική μελέτη**[P1]**. Η εξειδίκευση στο πρόβλημα μεταφοράς με ένα άλλο δάσος ξεκινήματος έγινε στην εργασία **[P2]**.

4.1 ΕΦΑΡΜΟΓΗ ΣΤΟ ΠΡΟΒΛΗΜΑ ΜΕΤΑΦΟΡΑΣ

4.1.1 Περιγραφή του Δάσους AKP για το πρόβλημα Μεταφοράς

Το δάσος AKP είναι πολύ εύκολο στην κατασκευή του. Έστω λοιπόν έχουμε να λύσουμε ένα $m \times n$ πρόβλημα Μεταφοράς και μας δίνονται ο $m \times n$ πίνακας δεδομένων c, το m-διάστατο διάνυσμα προσφοράς a και το n-διάστατο διάνυσμα προσφοράς b. Το δάσος AKP θα αποτελείται από μικρά δέντρα καθένα από τα οποία είναι ριζωμένα σε έναν κόμβο στήλη j. Κάθε δέντρο ρίζας j συμβολίζεται με T_j. Έτσι, αν F το δάσος AKP, θα ισχύει $F = \bigcup_{j=1}^{n} T_j$. Το δάσος AKP θα αποτελείται πάντα από m τόξα, αφού κάθε κόμβος γραμμή συνδέεται με τη ρίζα j ενός δέντρου T_j. Τα τόξα του δάσους F προσδιορίζονται ως εξής. Στην αρχή τίθεται

$$v_j(F) = 0,\ j = 1,...,n$$

Δηλαδή θέτουμε τις δυϊκές μεταβλητές των κόμβων στηλών ίσες με το μηδέν. Στη συνέχεια, προσδιορίζουμε τις δυϊκές μεταβλητές των κόμβων γραμμών $u_i(F),\ i = 1,...,m$ ως εξής. Τίθεται

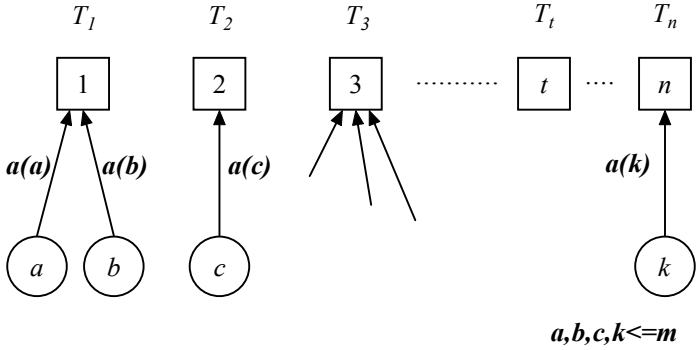
$$u_i(F) = \min\left\{c_{ij} : j = 1..n\right\},\ i = 1,...,m$$

Με την παραπάνω σχέση μπορούμε αμέσως να προσδιορίσουμε και τα τόξα $(i, j) \in F$. Συνεπώς, αν $u_i(F) = c_{it}$ τότε το τόξο $(i,t) \in F$. Ωστόσο, ο παραπάνω τρόπος δεν είναι ο μοναδικός για την εύρεση των τόξων που θα ανήκουν στο αρχικό δάσος. Μπορούν να χρησιμοποιηθούν και άλλες καλές δυϊκές ευρετικές συναρτήσεις.

Στη συνέχεια προσδιορίζουμε τις μεταβλητές απόφασης του δάσους ως εξής. Για κάθε κόμβο γραμμή i που συνδέεται με τον κόμβο στήλη j μπορούμε εύκολα να υπολογίσουμε τη μεταβλητή απόφασης $x_{ij}(F)$, ικανοποιώντας τις εξισώσεις των περιορισμών που εκφράζουν τη ροή στο δάσος. Συνεπώς τίθεται

$$x_{ij}(F) = a_i,\ \forall (i,j) \in F$$

Παρακάτω παρουσιάζουμε μια γενική εικόνα του δάσους ΑΚΡ για $m \times n$ προβλήματα μεταφοράς.



Εικόνα 4.0 – Ένα γενικό δάσος ΑΚΡ για *mxn* προβλήματα μεταφοράς

Παρατηρείστε ότι το δάσος μπορεί να περιέχει και απομονωμένους κόμβους. Επίσης παρατηρείστε ότι ο δάσος είναι δυϊκά εφικτό δηλαδή για κάθε $(i,j) \in F$ ισχύει

$$s_{ij}(F) = c_{ij} - u_i(F) - v_j(F) \geq 0$$

Παράδειγμα 4.1
Δίνεται ο πίνακας κόστους

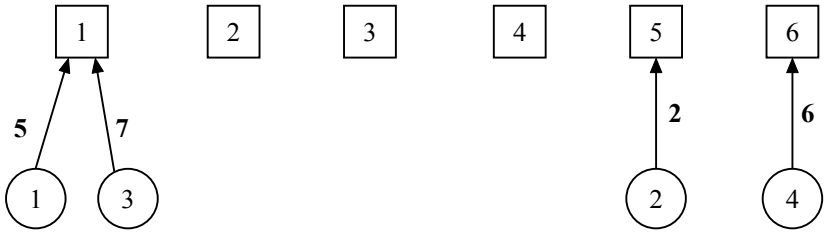
$$C = \begin{bmatrix} -5 & 49 & 28 & 20 & 47 & 43 \\ 19 & 62 & 32 & 61 & 0 & 13 \\ -2 & 63 & 14 & 46 & 28 & 53 \\ 1 & 50 & 41 & 26 & 59 & 0 \end{bmatrix}$$

, τα διανύσματα προσφοράς και ζήτησης $a = \begin{bmatrix} 5 & 2 & 7 & 6 \end{bmatrix}$ ***και*** $b = \begin{bmatrix} 2 & 3 & 7 & 2 & 1 & 5 \end{bmatrix}$ ***αντίστοιχα για ένα*** 4×6 ***πρόβλημα Μεταφοράς. Να προσδιορισθεί το δάσος ξεκινήματος ΑΚΡ και να υπολογισθούν τα στοιχεία*** $x_{ij}(F),\ (i,j) \in F$**,** $u_i(F),\ i = 1,...,m$**,** $v_j(F),\ j = 1,...,n$ ***και*** $s_{ij}(F),\ (i,j) \notin F$**.**

Λύση:
Έστω F το δάσος ΑΚΡ. Αρχικά θέτουμε $v_j(F) = 0,\ j = 1,...,n$. Στη συνέχεια, όπως είπαμε και στην περιγραφή του δάσους, θέτουμε $u_i(F) = \min\{c_{ij} : j = 1,...,n\},\ i = 1,...,m$. Εύκολα λοιπόν διαπιστώνουμε ότι το διάνυσμα $u = \begin{bmatrix} -5 & 0 & -2 & 0 \end{bmatrix}$. Το δάσος θα περιέχει $m = 4$ τόξα, τα (1,1), (2,5), (3,1) και (4,6). Αν θέσουμε τώρα $s_{ij}(F) = c_{ij} - u_i(F) - v_j(F)$, μπορούμε εύκολα να υπολογίσουμε τη μήτρα των μειωμένων κοστών. Έτσι θα έχουμε

$$S = \begin{bmatrix} 0 & 54 & 33 & 25 & 52 & 48 \\ 19 & 62 & 32 & 61 & 0 & 13 \\ 0 & 65 & 16 & 48 & 30 & 55 \\ 1 & 50 & 41 & 26 & 59 & 0 \end{bmatrix}$$

Τέλος, υπολογίζουμε τις τιμές των μεταβλητών απόφασης $x_{ij}(F)$ για κάθε τόξο $(i, j) \in F$ θέτοντας $x_{ij}(F) = a_i$, άρα θα έχουμε $x_{11}(F) = 5$, $x_{25}(F) = 2$, $x_{31}(F) = 7$ και $x_{46}(F) = 6$. Το τελικό δάσος ΑΚΡ του παραδείγματος φαίνεται στο παρακάτω σχήμα:



Εικόνα 4.1 – Το δάσος ΑΚΡ του προβλήματός μας

4.1.2 Περιγραφή του Αλγορίθμου

Ο αλγόριθμος ξεκινάει με το δάσος ΑΚΡ, το οποίο συμβολίζουμε με F. Συμβολίζουμε τα δέντρα που είναι ριζωμένα στον κόμβο στήλη j με T_j, $j = 1,...,n$. Τα δέντρα T_j χωρίζονται σε δύο διακριτά σύνολα τα F^S, F^D. Το σύνολο F^S ορίζεται ως εξής:

$$F^S = \left\{ T_j : \; b_j < \sum_{i \in T_j} a_i \right\}$$

Δηλαδή στο σύνολο F^S περιέχονται εκείνα τα δέντρα T_j, στην ρίζα j των οποίων εισέρχεται αυστηρά μεγαλύτερη ροή από τη ζήτηση του κόμβου στήλη j. Στα δέντρα $T_j \in F^S$ θα υπάρχει προσφορά $\sum_{i \in T_j} a_i - b_j > 0$. Αναλόγως το σύνολο F^D ορίζεται ως εξής:

$$F^D = \left\{ T_j : \; \sum_{i \in T_j} a_i \le b_j \right\}$$

Το σύνολο F^D περιέχει δηλαδή εκείνα τα δέντρα T_j, στην ρίζα j των οποίων εισέρχεται μικρότερη ή ίση ροή από τη ζήτηση του κόμβου στήλη j. Στα δέντρα $T_j \in F^D$ θα υπάρχει ζήτηση $\sum_{i \in T_j} a_i - b_j \le 0$.

Για να γίνει η περιγραφή του αλγορίθμου πιο πρακτική, προσθέτουμε στο δάσος F έναν τεχνητό κόμβο 0, τα τεχνητά τόξα $(r, 0)$, $(0, w)$ και κατασκευάζουμε πλέον το δέντρο T που αποτελείται από $m + n$ τόξα ώστε:

$$T = F \cup \{(r,0),\ \forall T_r \in F^S\} \cup \{(0,w),\ \forall T_w \in F^D\}$$

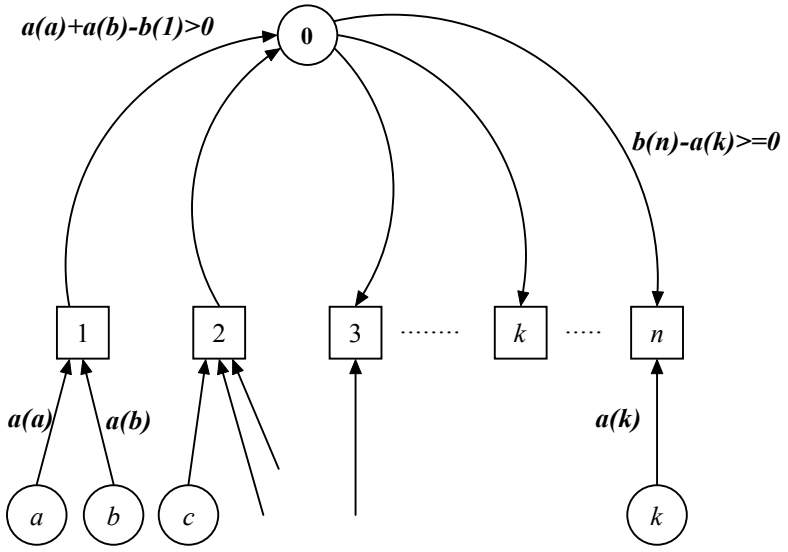
Παρατηρούμε ότι αν $T_j \in F^S$, το δέντρο T περιέχει τα τεχνητά τόξα $(j,0)$ και θα ισχύει

$$x_{j0} = \sum_{i \in T_j} a_i - b_j > 0$$

Αντιθέτως, αν ότι αν $T_j \in F^D$, το δέντρο T περιέχει τα τεχνητά τόξα $(0,j)$ και θα ισχύει

$$x_{0j} = b_j - \sum_{i \in T_j} a_i \geq 0$$

Συνεπώς, ένα δέντρο γενικής μορφής που χρησιμοποιεί ο αλγόριθμος είναι το παρακάτω:



Εικόνα 4.2 – Ένα δέντρο ΑΚΡ γενικής μορφής για το πρόβλημα Μεταφοράς

Παρατηρείστε ότι στο σχήμα 4.2 είναι $T_1, T_2 \in F^S$ και $T_3, T_k, T_n \in F^D$. Η εισερχόμενη και εξερχόμενη μεταβλητή προσδιορίζονται ως εξής. Έστω T το τρέχον δέντρο. Το εισερχόμενο τόξο προσδιορίζεται από τη σχέση:

$$\delta = s_{gh}(T) = \min\left\{ s_{ij}(T) : i \in F^S,\ j \in F^D \right\}$$

Στη συνέχεια, θα πρέπει να προσδιορίσουμε την εξερχόμενη μεταβλητή. Αρχικά, υπολογίζουμε τον κύκλο C που δημιουργεί η είσοδος του εισερχόμενου τόξου (g,h) στη βάση. Στη συνέχεια, υπολογίζουμε το σύνολο τόξων C^+ που αποτελείται από τα τόξα $(i,j) \in C$, τα οποία έχουν ίδια φορά με το εισερχόμενο τόξο (g,h). Το σύνολο C^- θα είναι το συμπληρωματικό του C^+ ως προς τον κύκλο C. Δηλαδή θα ισχύει:

$$C^- = C - C^+$$

Στη συνέχεια θέτουμε

$$\varepsilon = x_{k\ell} = \min\left\{ x_{ij}(T) : (i,j) \in C^- \right\}$$

Αν υπάρχουν περισσότερα από ένα τόξα που ικανοποιούν την παραπάνω σχέση, προσδιορίζουμε το σύνολο C' των επιλέξιμων τόξων, δηλαδή θέτουμε

$$C' = \left\{(i,j) \in C^- : x_{ij}(T) = \varepsilon\right\}$$

Το εξερχόμενο τόξο (k,ℓ) θα είναι τελικά αυτό που επιλέγει ο κανόνας του Cunningham, δηλαδή θα είναι το πρώτο επιλέξιμο τόξο που συναντάμε όταν διαγράφουμε τον κύκλο ξεκινώντας από την ρίζα προς τη φορά του εισερχόμενου τόξου **[Cun]**.

Το νέο δέντρο που κατασκευάζεται είναι το δέντρο $T' = T \cup (g,h) \sim (k,\ell)$. Οι τιμές των μεταβλητών απόφασης ανανεώνονται σύμφωνα με τον τύπο

$$x_{ij}(T') = \begin{cases} x_{ij}(T) + \varepsilon & ,\forall (i,j) \in C^+ \\ x_{ij}(T) - \varepsilon & ,\forall (i,j) \in C^- \end{cases}$$

Μεταβάλλονται μόνο οι μεταβλητές s_{ij} τόξων (i,j) που προσπίπτουν σε κόμβους που ανήκουν στο αποκομμένο δέντρο $T*$. Ο αλγόριθμος σταματά όταν $F^S = \varnothing$, κάτι που επιτυγχάνεται σίγουρα.

4.1.3 Βηματική Περιγραφή του Αλγορίθμου

Στη συνέχεια θα παρουσιάσουμε τον αλγόριθμο σε μορφή βημάτων και βασισμένοι στα βήματα του αλγορίθμου θα λύσουμε ένα πρόβλημα.

ΒΗΜΑ 0:*(Καθορισμός αρχικών μεταβλητών)*
Υπολόγισε το δάσος F ΑΚΡ. Καθόρισε τα σύνολα F^S και F^D με τη μέθοδο που περιγράψαμε. Κατασκεύασε το αρχικό δέντρο T προσθέτοντας έναν τεχνητό κόμβο 0 και τα τεχνητά τόξα $(j,0)$ για κάθε $j \in F^S$ και $(0,j)$ για κάθε $j \in F^D$.

ΒΗΜΑ 1:*(Έλεγχος βελτιστότητας)*

βέλτιστο και ο αλγόριθμος τερματίζει, αλλιώς πήγαινε στο βήμα 2.

ΒΗΜΑ 2:*(Επιλογή εισερχομένου τόξου)*
Υπολόγισε το εισερχόμενο τόξο (g,h) ελέγχοντας τα μειωμένα κόστη των μη βασικών τόξων με βάση τον τύπο $\delta = s_{gh} = \min\left\{s_{ij}(T) : i \in F^S, j \in F^D\right\}$.

ΒΗΜΑ 3:*(Επιλογή εξερχόμενου τόξου)*
Υπολόγισε τα σύνολα τόξων C^+, C^- και καθόρισε τη μεταβλητή ε, θέτοντας $\varepsilon = \min\left\{x_{ij}(T) : (i,j) \in C^-\right\}$.Βρες το σύνολο C' των επιλέξιμων τόξων, θέτοντας $C' = \left\{(i,j) \in C^- : \ x_{ij}(T) = \varepsilon\right\}$. Επέλεξε το εξερχόμενο τόξο $(k,\ell) \in C^-$ με τον κανόνα του Cunningham, δηλαδή το εξερχόμενο τόξο θα είναι το πρώτο επιλέξιμο τόξο $(r,t) \in C^-$ που συναντάμε κατά την διάσχιση του κύκλου C, ξεκινώντας από τη ρίζα του δέντρου.

ΒΗΜΑ 4:*(Ανανέωση των μεταβλητών)*

Θέσε $x_{ij}(T') = x_{ij}(T) - \varepsilon$ για κάθε τόξο $(i,j) \in C^-$ και αντιστοίχως $x_{ij}(T') = x_{ij}(T) + \varepsilon$ για κάθε τόξο $(i,j) \in C^+$. Στη συνέχεια ανανέωσε τα σύνολα F^S, F^D. Υπολόγισε το δέντρο T^* που είναι το δέντρο που αποκόβεται από τη ρίζα όταν αφαιρείται το εξερχόμενο τόξο. Αν $h \in T^*$ θέσε $q = -\delta > 0$, αλλιώς θέσε $q = \delta < 0$. Στη συνέχεια εάν ο κόμβος $b \in T^*$ είναι κόμβος-γραμμή θέσε $s_{bj}(T') = s_{bj}(T) - q$, $j = 1,...,n$ αλλιώς (δηλαδή ο κόμβος $b \in T^*$ είναι κόμβος-στήλη) θέσε $s_{ib}(T') = s_{ib}(T) + q$, $i = 1,...,m$. (Ανανεώνονται οι τιμές s_{ij} των τόξων (i,j) που προσπίπτουν σε κόμβους που ανήκουν στο δέντρο T^*)

ΒΗΜΑ 5:*(Ανανέωση του τρέχοντος δέντρου)*

Στη συνέχεια θα παρουσιάσουμε την ακριβή επίλυση ενός παραδείγματος. Θα χρησιμοποιήσουμε τα δεδομένα που χρησιμοποιήσαμε στο παράδειγμα 4.1.

Παράδειγμα 4.2

Δίνονται το διάνυσμα προσφοράς $a = [5 \quad 2 \quad 7 \quad 6]$***, το διάνυσμα ζήτησης*** $b = [2 \quad 3 \quad 7 \quad 2 \quad 1 \quad 5]$***, και ο πίνακας κόστους***

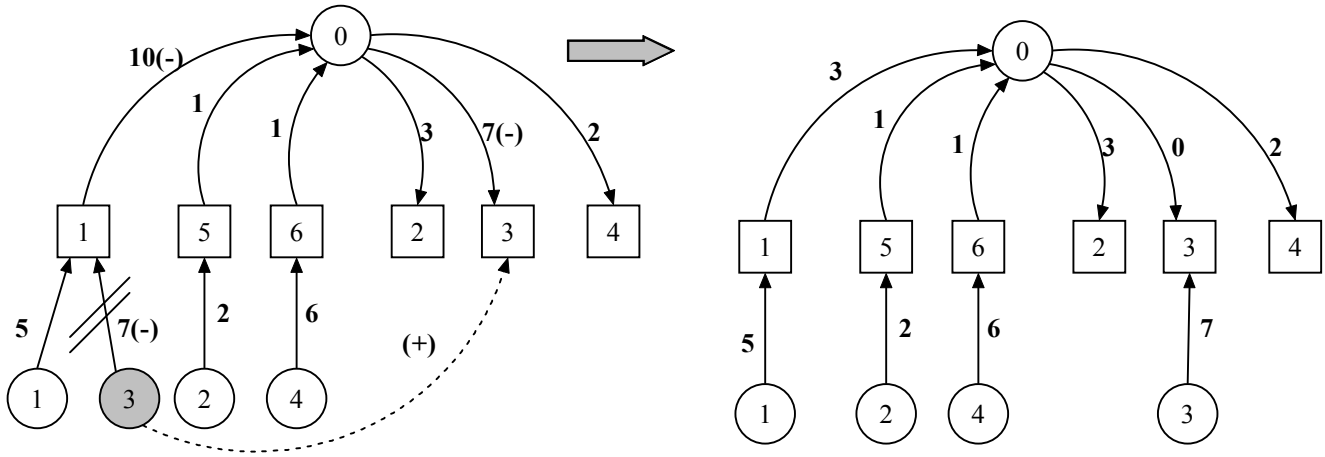
$$C = \begin{bmatrix} -5 & 49 & 28 & 20 & 47 & 43 \\ 19 & 62 & 32 & 61 & 0 & 13 \\ -2 & 63 & 14 & 46 & 28 & 53 \\ 1 & 50 & 41 & 26 & 59 & 0 \end{bmatrix}$$

για ένα 4×6 ***πρόβλημα Μεταφοράς. Να λυθεί με τον αλγόριθμο Παπαρρίζου και να χρησιμοποιηθεί ως αρχική εφικτή λύση το δάσος ξεκινήματος ΑΚΡ.***

Λύση:

Αρχικά, πρέπει να προσδιορίσουμε το δάσος ΑΚΡ (σχήμα 4.1).

Επανάληψη 1



5	54	33	25	52	48	
19	62	32	61	**2**	13	
1	65	16	48	30	55	**-16**
1	50	41	26	59	**6**	

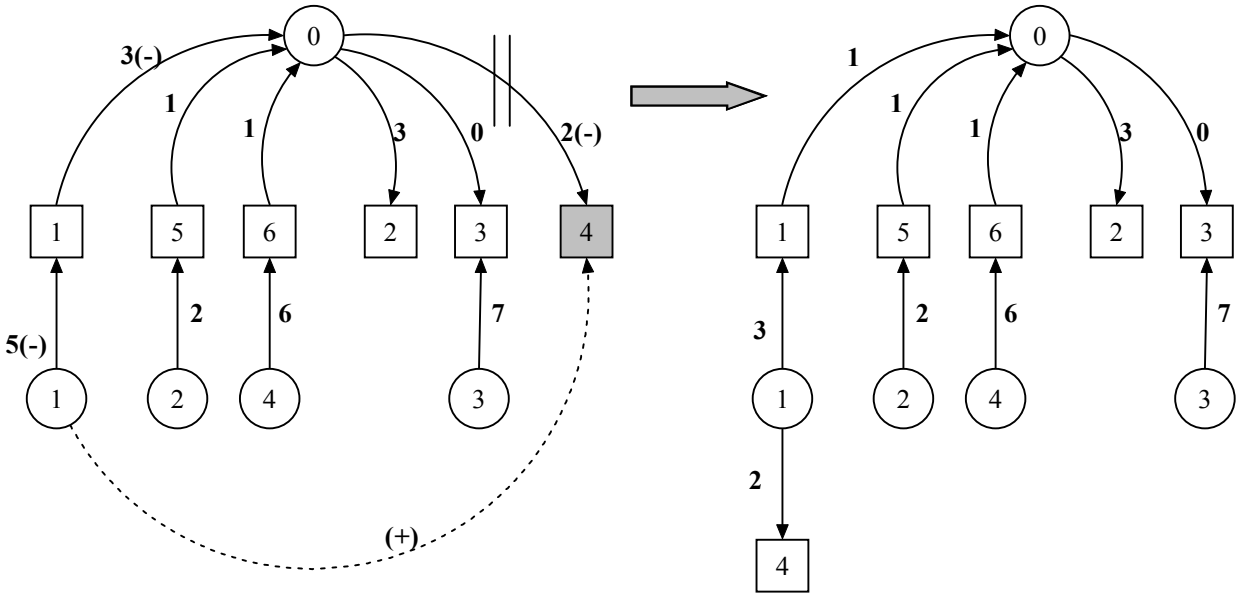
5	54	33	25	52	48
19	62	32	61	**2**	13
-16	49	**7**	32	14	39
1	50	41	26	59	**6**

Εικόνα 4.3 – Η πρώτη επανάληψη του αλγορίθμου

- Είναι $F^S = \{1,2,3,4\}$, $F^D = \{2,3,4\}$, $\delta = s_{gh} = \min\{s_{ij}(T) : i \in F^S, j \in F^D\} = s_{33} = 16$,[1] συνεπώς $(g,h) = (3,3)$.
- Τα σύνολα C^+, C^-, φαίνονται στο σχήμα 4.3. Χρησιμοποιώντας τη γνωστή σχέση $\varepsilon = x_{k\ell} = \min\{x_{ij}(T) : (i,j) \in C^-\}$, προσδιορίζουμε το εξερχόμενο τόξο (k,ℓ). Έτσι $\varepsilon = 7$ και $(k,\ell) = (3,1)$.
- Η αναλυτική ανανέωση των μεταβλητών x_{ij}, s_{ij} φαίνεται στο σχήμα 4.3

Επανάληψη 2

- Είναι $F^S = \{1,2,4\}$, $F^D = \{2,3,4\}$, $\delta = s_{gh} = \min\{s_{ij}(T) : i \in F^S, j \in F^D\} = s_{14} = 25$, συνεπώς $(g,h) = (1,4)$.
- Τα σύνολα C^+, C^-, φαίνονται στο σχήμα 4.4. Χρησιμοποιώντας τη γνωστή σχέση $\varepsilon = x_{k\ell} = \min\{x_{ij}(T) : (i,j) \in C^-\}$, προσδιορίζουμε το εξερχόμενο τόξο (k,ℓ). Έτσι $\varepsilon = 2$ και $(k,\ell) = (0,4)$.
- Η αναλυτική ανανέωση των μεταβλητών x_{ij}, s_{ij} φαίνεται στο σχήμα 4.4



5	54	33	25	52	48
19	62	32	61	**2**	13
-16	49	**7**	32	14	39
1	50	41	26	59	**6**

-25

3	54	33	**2**	52	48
19	62	32	36	**2**	13
-16	49	**7**	7	14	39
1	50	41	1	59	**6**

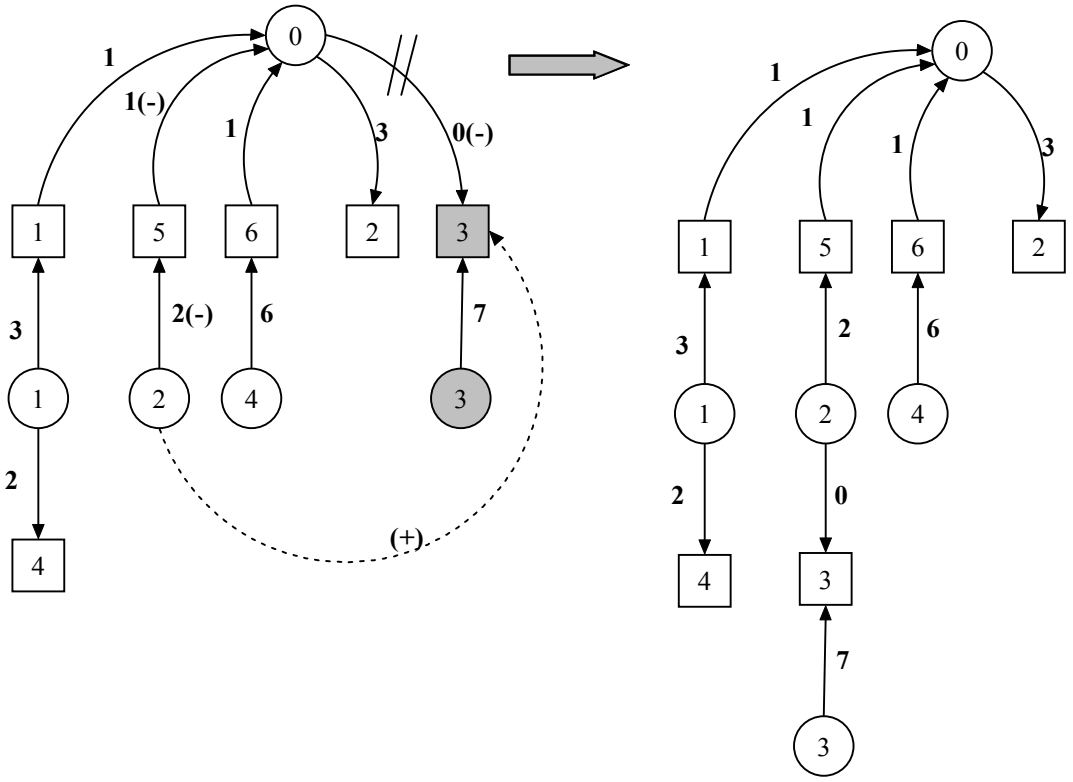
Εικόνα 4.4 – Η δεύτερη επανάληψη του αλγορίθμου

Επανάληψη 3

- Είναι $F^S = \{1,2,4\}$, $F^D = \{2,3\}$, $\delta = s_{gh} = \min\{s_{ij}(T) : i \in F^S, j \in F^D\} = s_{23} = 32$, συνεπώς $(g,h) = (2,3)$.

[1] Τα σύνολα F^S, F^D προς χάριν ευκολίας περιέχουν κόμβους γραμμές και στήλες αντίστοιχα.

- Τα σύνολα C^+, C^-, φαίνονται στο σχήμα 4.5. Χρησιμοποιώντας τη γνωστή σχέση $\varepsilon = x_{k\ell} = \min\left\{x_{ij}(T) : (i,j) \in C^-\right\}$, προσδιορίζουμε το εξερχόμενο τόξο (k,ℓ). Έτσι $\varepsilon = 0$ και $(k,\ell) = (0,3)$.
- Η αναλυτική ανανέωση των μεταβλητών x_{ij}, s_{ij} φαίνεται στο σχήμα 4.5



3	54	33	**2**	52	48	
19	62	32	36	**2**	13	
-16	49	**7**	7	14	39	**+32**
1	50	41	1	59	**6**	
		-32				

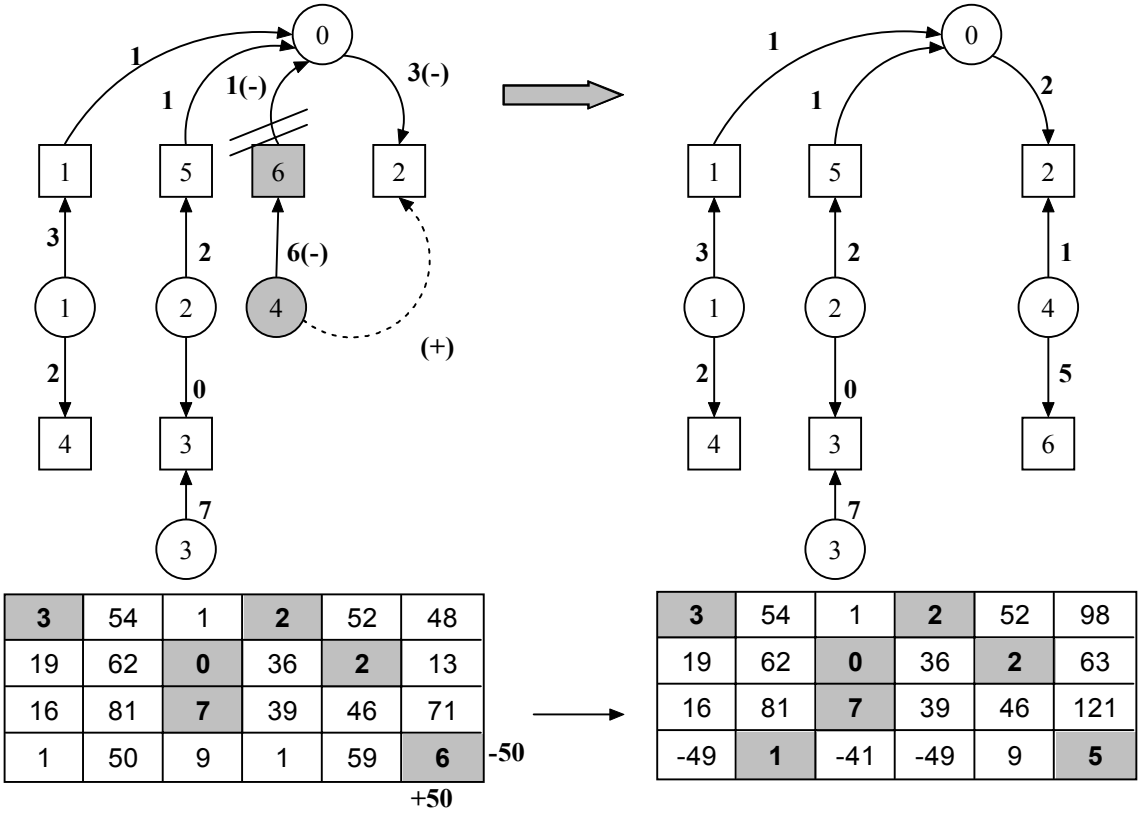
3	54	1	**2**	52	48
19	62	**0**	36	**2**	13
16	81	**7**	39	46	71
1	50	9	1	59	**6**

Εικόνα 4.5 - Η τρίτη επανάληψη του αλγορίθμου

Επανάληψη 4

- Είναι $F^S = \{1,2,3,4\}$, $F^D = \{2\}$, $\delta = s_{gh} = \min\left\{s_{ij}(T) : i \in F^S, j \in F^D\right\} = s_{42} = 50$, συνεπώς $(g,h) = (4,2)$.
- Τα σύνολα C^+, C^-, φαίνονται στο σχήμα 4.6. Χρησιμοποιώντας τη γνωστή σχέση $\varepsilon = x_{k\ell} = \min\left\{x_{ij}(T) : (i,j) \in C^-\right\}$, προσδιορίζουμε το εξερχόμενο τόξο (k,ℓ). Έτσι $\varepsilon = 1$ και $(k,\ell) = (6,0)$.
- Η αναλυτική ανανέωση των μεταβλητών x_{ij}, s_{ij} φαίνεται στο σχήμα 4.6.[2]

[2] Ανανεώνονται μόνο οι τιμές μειωμένων κοστών των τόξων που έχουν έναν μόνο κόμβο στο δέντρο T^*. Το δέντρο T^* στην τέταρτη επανάληψη θα αποτελείται από τον κόμβο στήλη 6 και τον κόμβο γραμμή 4. Συνεπώς, προσθέτουμε +50 στις στη στήλη 6 του πίνακα s_{ij} ενώ προσθέτουμε -60 στις γραμμή 4 του πίνακα s_{ij}. Η ανανέωση αυτή γίνεται με τη μέθοδο που αναφέρθηκε στο βήμα 4 της περιγραφής του αλγορίθμου. Εξ'άλλου,

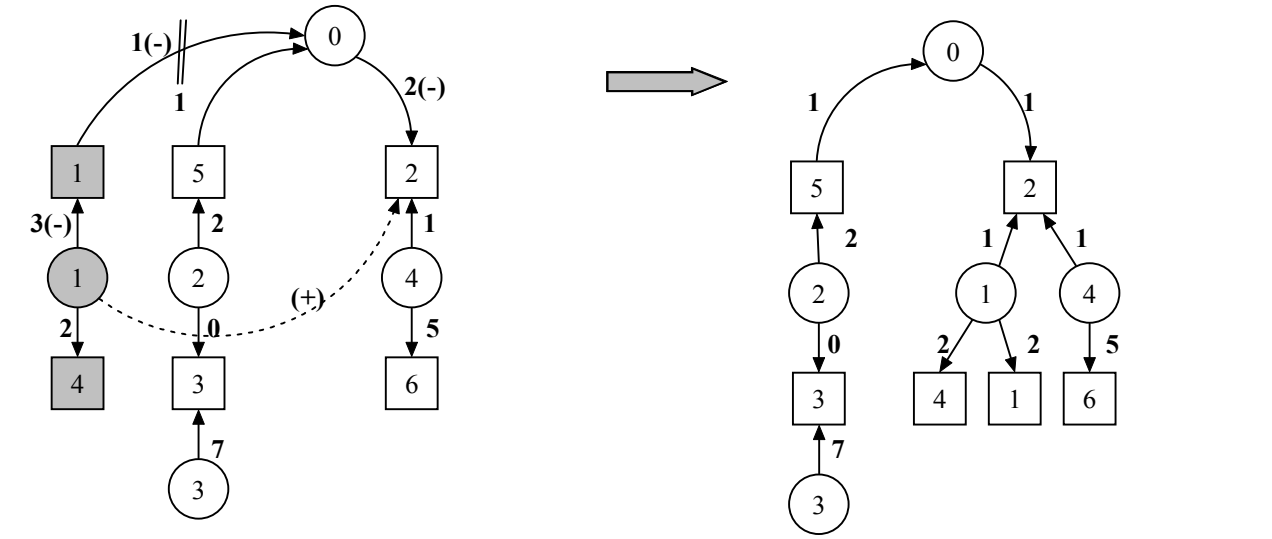


3	54	1	**2**	52	48	
19	62	**0**	36	**2**	13	
16	81	**7**	39	46	71	
1	50	9	1	59	**6**	**-50**
					+50	

3	54	1	**2**	52	98
19	62	**0**	36	**2**	63
16	81	**7**	39	46	121
-49	**1**	-41	-49	9	**5**

Εικόνα 4.6 – Η τέταρτη επανάληψη του αλγορίθμου

Επανάληψη 5



3	54	1	**2**	52	98	
19	62	**0**	36	**2**	63	
16	81	**7**	39	46	121	
-49	**1**	-41	-49	9	**5**	**-54**
+54			**+54**			

2	**1**	-53	**2**	-2	44
73	62	**0**	90	**2**	63
70	81	**7**	93	46	121
5	**1**	-41	5	9	**5**

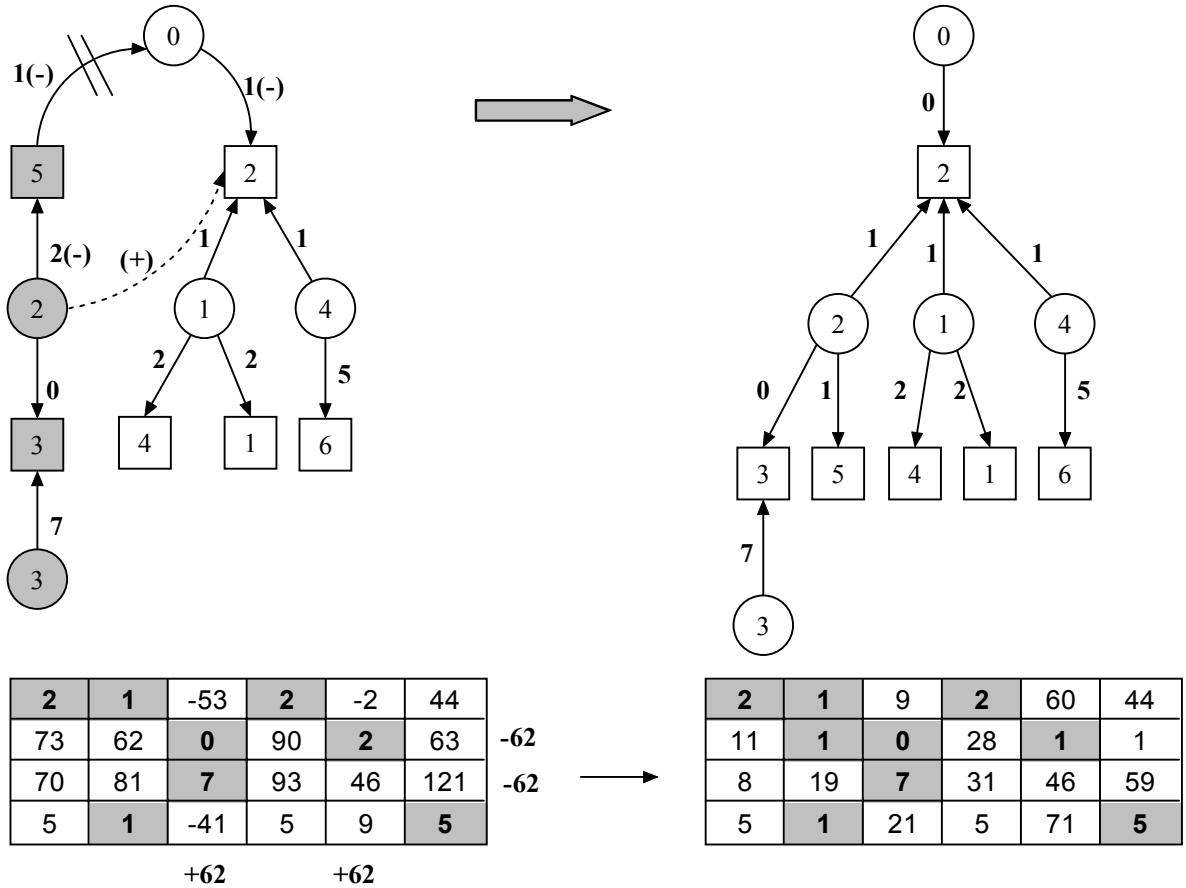
Εικόνα 4.7 – Η πέμπτη επανάληψη του αλγορίθμου

μπορούμε εύκολα να καταλάβουμε που πρέπει να προσθέσουμε και αντίστοιχα να αφαιρέσουμε δ, αν λάβουμε υπ' όψιν το γεγονός ότι το μειωμένο κόστος του εισερχόμενου τόξου (g,h) πρέπει να μηδενιστεί μετά το πέρας της επανάληψης, αφού το μειωμένο κόστος κάθε βασικού τόξου είναι πάντα ίσο με το μηδέν.

- Είναι $F^S=\{1,2,3\}$, $F^D=\{2,6\}$, $\delta=s_{gh}=\min\{s_{ij}(T): i\in F^S, j\in F^D\}=s_{12}=54$, συνεπώς $(g,h)=(1,2)$.
- Τα σύνολα C^+, C^-, φαίνονται στο σχήμα 4.7. Χρησιμοποιώντας τη γνωστή σχέση $\varepsilon=x_{k\ell}=\min\{x_{ij}(T):(i,j)\in C^-\}$, προσδιορίζουμε το εξερχόμενο τόξο (k,ℓ). Έτσι $\varepsilon=1$ και $(k,\ell)=(1,0)$.
- Η αναλυτική ανανέωση των μεταβλητών x_{ij}, s_{ij} φαίνεται στο σχήμα 4.7

Επανάληψη 6

- Είναι $F^S=\{2,3\}$, $F^D=\{2,4,6\}$, $\delta=s_{gh}=\min\{s_{ij}(T): i\in F^S, j\in F^D\}=s_{22}=62$, συνεπώς $(g,h)=(2,2)$.
- Τα σύνολα C^+, C^-, φαίνονται στο σχήμα 4.7. Χρησιμοποιώντας τη γνωστή σχέση $\varepsilon=x_{k\ell}=\min\{x_{ij}(T):(i,j)\in C^-\}$, προσδιορίζουμε το εξερχόμενο τόξο (k,ℓ). Έτσι $\varepsilon=1$ και $(k,\ell)=(5,0)$.
- Η αναλυτική ανανέωση των μεταβλητών x_{ij}, s_{ij} φαίνεται στο σχήμα 4.8



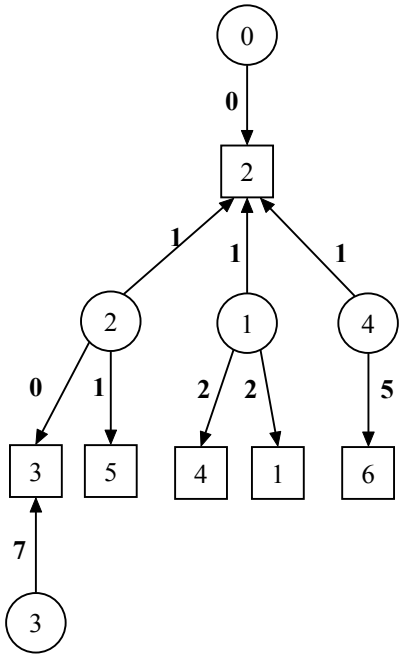
2	**1**	-53	**2**	-2	44	
73	62	**0**	90	**2**	63	**-62**
70	81	**7**	93	46	121	**-62**
5	**1**	-41	5	9	**5**	
		+62		**+62**		

2	**1**	9	**2**	60	44
11	**1**	**0**	28	**1**	1
8	19	**7**	31	46	59
5	**1**	21	5	71	**5**

Εικόνα 4.8 – Η έκτη επανάληψη του αλγορίθμου

Παρατηρούμε ότι στο τελευταίο δέντρο που παράχθηκε είναι $F^S=\varnothing$. Συνεπώς, το δέντρο είναι βέλτιστο και ο αλγόριθμος τερματίζει. Παρατηρείστε ότι το τελευταίο δέντρο περιέχει τεχνητά τόξα που είναι εκφυλισμένα. Αυτό είναι κάτι που το εγγυάται ο αλγόριθμος ότι θα συμβεί **[P2]**. Στην εικόνα 4.9 βλέπουμε το βέλτιστο δέντρο του αλγορίθμου. Παρατηρούμε ότι η ύπαρξη του ουδέτερου κόμβου 0 δεν είναι πλέον απαραίτητη, αφού συνδέεται με ένα

τεχνητό εκφυλισμένο τόξο με το υπόλοιπο δέντρο. Η τιμή της αντικειμενικής συνάρτησης είναι $z^* = \sum_{(i,j)\in T} c_{ij}x_{ij} = 289$.



Εικόνα 4.9 – Το βέλτιστο δέντρο του προβλήματος

4.1.4 Προγραμματισμός του Αλγορίθμου

Αλγόριθμος Τ3
Ο παρακάτω αλγόριθμος λύνει το ισοζυγισμένο πρόβλημα μεταφοράς , χρησιμοποιώντας το αλγόριθμο Παπαρρίζου με ξεκίνημα το δάσος ΑΚΡ.
Είσοδος: *Το διάνυσμα προσφοράς* **A(mx1)**,*το διάνυσμα ζήτησης* **B(nx1)** *και ο πίνακας κόστους* **C(mxn)**.
Έξοδος : *Η λύση του προβλήματος Μεταφοράς* **X(mxn)**.

```
1.  xv(1:m+n)=0;
2.  [X,S]=AKPtree(A,C,m,n);
3.  p=InitializeNodeFather(X,m,n);
4.  d=InitializeNodeDepth(p,m,n);
5.  [Fs,Fd]=SupplyDemandSets(X,m,n,p,xv,B);
6.  Ematrix=Initialize(S,Fs,Fd,m,n);
7.  while count<m+n
8.    [Fs,Fd]=SupplyDemandSets(X,m,n,p,xv,B);
9.    count=0;
10.   for q=1:m+n
11.     if Fs(q)==0
12.       count=count+1;
13.     end
14.   end
15.   if count<m+n
16.     t=NodeDirection(Fs,Fd,m,n,p);
17.     xv=Vectorx(B,p,X,m,n,xv,A);
18.     [min,g,h,f,Circle]=EnteringArc(S,Fs,Fd,m,n,p,Ematrix);
19.     [e,k,l,Cplus,Cminus]=LeavingArc(g,h,p,m,n,d,t,xv);
20.     [e1,e2,f1,z,Tcut]=Steam(p,g,h,k,l,d,m,n);
21.     [S,X,Ematrix]=UpdateVariables(min,e,g,h,...,Ematrix,Fs,Fd);
22.     d=UpdateNodeDepth(p,m,n,Tcut,z,d,e1,e2,f1);
23.     p=UpdateNodeFather(p,z,e2,f1);
24.   end
25. end
```

Αλγόριθμος 4.1.1 – Ο κώδικας υλοποίησης του αλγόριθμου Παπαρρίζου με ξεκίνημα το δάσος ΑΚΡ

Παραπάνω βλέπουμε όλες τις συναρτήσεις που χρησιμοποιήσαμε για τον προγραμματισμό του αλγορίθμου. Υλοποιώντας αυτές τις συναρτήσεις σωστά σε οποιαδήποτε γλώσσα προγραμματισμού, μπορεί εύκολα κάποιος να προγραμματίσει τον αλγόριθμο με ανάλογο τρόπο. Στις πρώτες γραμμές του κώδικα αρχικοποιούνται ορισμένες μεταβλητές απαραίτητες για την εκτέλεση του αλγορίθμου. Στη συνέχεια ελέγχεται αν το σύνολο F^S είναι το κενό σύνολο έτσι ώστε να πάρουμε μια απόφαση για το αν πρέπει να συνεχιστεί η εκτέλεση του αλγορίθμου. Στις επόμενες γραμμές του κώδικα ακολουθείται η ίδια μεθοδολογία που ακολουθήθηκε και στους άλλους αλγορίθμους, δηλαδή αρχικά υπολογίζεται το εισερχόμενο τόξο (g,h) , μετά προσδιορίζεται το εξερχόμενο τόξο (k,ℓ) και στη συνέχεια ανανεώνονται οι τιμές των διάφορων μεταβλητών που χρησιμοποιεί ο αλγόριθμος.

- **[X , S] = AKPtree(A, C, m, n);**

Αλγόριθμος AKPTree
Ο αλγόριθμος που προσδιορίζει το δάσος ξεκινήματος AKP.
Είσοδος: *Το διάνυσμα προσφοράς* **A(mx1)**, *ο πίνακας κόστους* **C(mxn)**, **m** , **n** .
Έξοδος : *Το εφικτό δέντρο* **X(mxn)** *και ο αντίστοιχος πίνακας μειωμένων κοστών* **S(mxn)** .

```
1.  function [X,S]=AKPtree(A,C,m,n)
2.  for i=1:n
3.    V(i)=0;
4.  end
5.  for i=1:m
6.    for j=1:n
7.      X(i,j)=0;
8.      S(i,j)=0;
9.    end
10. end
11. minimum=inf;
12. for i=1:m
13.   for j=1:n
14.     if C(i,j)<minimum
15.       minimum=C(i,j);
16.       t=j;
17.     end
18.   end
19.   U(i)=minimum;
20.   X(i,t)=A(i);
21.   minimum=inf;
22. end
23. for i=1:m
24.   for j=1:n
25.     if X(i,j)==0
26.       X(i,j)=inf;
27.     end
28.     S(i,j)=C(i,j)-U(i)-V(j);
29.   end
30. end
```

Αλγόριθμος 4.1.2 – Ο αλγόριθμος υπολογισμού του δάσους AKP

Αν παρατηρήσουμε προσεχτικά τον παραπάνω κώδικα, βλέπουμε ότι συμβαδίζει απόλυτα με την περιγραφή του δάσους AKP. Συγκεκριμένα, αρχικά μηδενίζονται οι δυϊκές μεταβλητές των κόμβων στηλών. Στη συνέχεια, ψάχνουμε για το ελάχιστο στοιχείο σε κάθε γραμμή του πίνακα κόστους και με αυτόν τον τρόπο καθορίζουμε τα βασικά τόξα του δάσους. Μετά (γραμμή 28), υπολογίζονται τα μειωμένα κόστη των μη τόξων (i,j) με βάση το γνωστό τύπο

$$s_{ij}(T) = c_{ij} - u_i(T) - v_j(T)$$

- **Ematrix = InitializeEm(S, Fs, Fd, m, n)**

Στη συνέχεια, θα παρουσιάσουμε έναν νέο τρόπο για την επιλογή του εισερχομένου τόξου. Αυτός ο τρόπος προτείνεται στην εργασία **[P1]** που αναλύει τον ίδιο αλγόριθμο για το πρόβλημα Αντιστοίχησης. Ο αλγόριθμος μπορεί να αποδειχθεί ότι αποτελείται από στάδια (*levels*). Ένα στάδιο του αλγορίθμου είναι το σύνολο των επαναλήψεων κατά τη διάρκεια των οποίων το σύνολο F^S αυξάνεται, δηλαδή κόβονται τόξα $(k,\ell) \in F^D$ και στην τελευταία επανάληψη των οποίων κόβεται ένα τόξο $(k,\ell) \in F^S$ με αποτέλεσμα να αυξάνει πάλι το σύνολο F^D. Αυτή την αλληλουχία σταδίων μπορούμε να την εκμεταλλευτούμε και να εφαρμόσουμε έναν πιο «κομψό» τρόπο επιλογής του εισερχόμενου τόξου. Για να το επιτύχουμε αυτό, χρησιμοποιούμε έναν πίνακα **Ematrix** διαστάσεων $2 \times n$. Κάθε στήλη του πίνακα αναφέρεται σε ακριβώς έναν κόμβο στήλη του συνόλου F^D. Στο στοιχείο $(1, j)$ του πίνακα **Ematrix** αποθηκεύουμε το κόμβο γραμμή i για τον οποίο ισχύει $s_{ij} = \min\{s_{aj} : a \in F^S\}$. Στο στοιχείο $(2, j)$ του πίνακα αποθηκεύουμε το αντίστοιχο μειωμένο κόστος s_{ij}. Για τους κόμβους στήλες $t \notin F^D$ θέτουμε την αντίστοιχη στήλη του πίνακα **Ematrix** ίση με άπειρο. Για να κατανοηθεί καλύτερα η δομή του πίνακα **Ematrix,** ας κοιτάξουμε πάλι την πέμπτη επανάληψη του αλγορίθμου στο παράδειγμα που λύσαμε(εικόνα 4.7). Παρατηρούμε ότι $F^S = \{1,2,3\}$ και $F^D = \{2,6\}$. Επίσης ο πίνακας μειωμένων κοστών είναι

$$s = \begin{bmatrix} 0 & 54 & 1 & 0 & 52 & 98 \\ 19 & 62 & 0 & 36 & 0 & 63 \\ 16 & 81 & 0 & 39 & 46 & 121 \\ -49 & 0 & -41 & -49 & 9 & 0 \end{bmatrix}$$

Συνεπώς, ο πίνακας **Ematrix** θα έχει πεπερασμένες τιμές μόνο στις στήλες 2,6. Έτσι για την πέμπτη επανάληψη θα είναι:

$$\textbf{Ematrix} = \begin{bmatrix} \infty & 1 & \infty & \infty & \infty & 2 \\ \infty & 54 & \infty & \infty & \infty & 63 \end{bmatrix}$$

Στη συνέχεια παραθέτουμε τον κώδικα για την αρχικοποίηση του πίνακα **Ematrix**. Καλείται μόνο μια φορά στη αρχή.

Αλγόριθμος InitializeEm
Ο αλγόριθμος που αρχικοποιεί τον πίνακα **Ematrix**
Είσοδος: S(mxn), Fs(), Fd(), m, n
Έξοδος : Ematrix(2xn)

```
1.  function Ematrix=Initialize(S,Fs,Fd,m,n)
2.  for (i=1:2)
3.    for (j=1:n)
4.      Ematrix(i,j)=inf;
5.    end
6.  end
7.  j=1;
8.  m9=inf;
9.  while (Fd(j)~=0)
10.   i=1;
```

```
11.  m9=inf;
12.  while (Fs(i)~=0)
13.    if Fs(i)<=m & Fd(j)>m
14.      if S(Fs(i),Fd(j)-m)<m9
15.        m9=S(Fs(i),Fd(j)-m);
16.        Ematrix(2,Fd(j)-m)=S(Fs(i),Fd(j)-m);
17.        Ematrix(1,Fd(j)-m)=Fs(i);
18.      end
19.    end
20.    i=i+1;
21.  end
22.  j=j+1;
23.end
```

Αλγόριθμος 4.1.3 – Ο αλγόριθμος αρχικοποίησης του πίνακα *Ematrix*.

Παρατηρείστε ότι για να καθοριστεί τώρα το εισερχόμενο τόξο, χρειάζεται απλώς να βρεθεί το ελάχιστο στοιχείο της πρώτης γραμμής του πίνακα **Ematrix**. Επίσης, χρειάζεται να βρεθεί ένας τρόπος για την άμεση ανανέωση του πίνακα **Ematrix**, πράγμα που υλοποιείται με επιτυχία παρακάτω.

- **[S, X, Ematrix] =UpdateVariables([3])**

Στη συνέχεια, παρουσιάζουμε τον κώδικα για την ανανέωση των μεταβλητών του αλγορίθμου. Ιδιαίτερο ενδιαφέρον παρουσιάζουν οι τελευταίες γραμμές του κώδικα, όπου γίνεται η ανανέωση του πίνακα **Ematrix**.

Αλγόριθμος UpdateVariables
Ανανέωση των μεταβλητών του αλγορίθμου
Είσοδος: [3]
Έξοδος : S(mxn), X(mxn), Ematrix(2xn)

```
1.  i=1;
2.  while Cminus(i)~=0
3.    if (t(Cminus(i))==1)&(p(Cminus(i))~=0)
4.      X(Cminus(i),p(Cminus(i))-m)=X(Cminus(i),p(Cminus(i))-m)-e;
5.    elseif (t(Cminus(i))==0)&(p(Cminus(i))~=0)
6.      X(p(Cminus(i)),Cminus(i)-m)=X(p(Cminus(i)),Cminus(i)-m)-e;
7.    end
8.    i=i+1;
9.  end
10. i=1;
11. while Cplus(i)~=0
12.   if (t(Cplus(i))==1)
13.     X(Cplus(i),p(Cplus(i))-m)=X(Cplus(i),p(Cplus(i))-m)+e;
14.   elseif t(Cplus(i))==0
15.     X(p(Cplus(i)),Cplus(i)-m)=X(p(Cplus(i)),Cplus(i)-m)+e;
16.   end
17. i=i+1;
18. end
19. if (k~=0)&(l~=0)
20.   X(k,l)=inf;
21.   X(g,h)=e;
22. elseif k==0
23.   xv(l+m)=inf;
24.   X(g,h)=e;
25. elseif l==0
26.   xv(k+m)=inf
27.   X(g,h)=e;
28. end
29. for i=1:length(Tcut)
```

[3] **min ,e, g, h, k, l, Tcut(), Cminus(), Cplus(), m, n, t(m+n), p(m+n), S(mxn), X(mxn), Ematrix(2xn), Fs(), Fd()**

```
  if h+m==Tcut(i)
    min=-min;
  end
end
for i=1:length(Tcut)
  if Tcut(i)<=m
    for j=1:n
      S(Tcut(i),j)=S(Tcut(i),j)-min;
    end
  else
    for j=1:m
      S(j,Tcut(i)-m)=S(j,Tcut(i)-m)+min;
    end
  end
end
for j=1:n
  if Ematrix(1,j)~=inf
    Ematrix(2,j)=S(Ematrix(1,j),j);
  end
end
i=1;
j=1;
q=1;
for i=1:length(Tcut)
  if Tcut(i)<=m
    tcr(j)=Tcut(i);
    j=j+1;
  else
    tcc(q)=Tcut(i);
    q=q+1;
  end
  i=i+1;
end
Fs2=[];
if any(l+m==Fs)
  i=1;
  c=1;
  while Fs(i)~=0
    if Fs(i)<=m
      if ~any(Fs(i)==tcr)
        Fs2(c)=Fs(i);
        c=c+1;
      end
    end
    i=i+1;
  end
  if c>1
    for j=1:n
      if Ematrix(1,j)~=inf | any(j+m==tcc)
        m3=inf;
        for i=Fs2
          if S(i,j)<m3
            m3=S(i,j);
            Ematrix(2,j)=S(i,j);
            Ematrix(1,j)=i;
          end
        end
      end
     end
  end
  else
    Ematrix(1,h)=inf;
    Ematrix(2,h)=inf;
    for i=tcc
      Ematrix(1,i-m)=inf;
      Ematrix(2,i-m)=inf;
    end
    for i=1:n
```

```
98.       if Ematrix(1,i)~=inf
99.       for kk=tcr
100.         if S(kk,i)<Ematrix(2,i)
101.           Ematrix(2,i)=S(kk,i);
102.           Ematrix(1,i)=kk;
103.         end
104.         end
105.         end
106.         end
107.         end
```

Αλγόριθμος 4.1.4 – Ο αλγόριθμος ανανέωσης των μεταβλητών του αλγορίθμου.

Η ανανέωση του πίνακα **Ematrix** γίνεται από τη γραμμή 76 και μετά. Η διαδικασία είναι τέτοια έτσι ώστε να εξασφαλίζεται ότι ο πίνακάς πάντα θα έχει πεπερασμένες τιμές στις στήλες j που ανήκουν στο σύνολο F^D, οι οποίες πρέπει να αντιστοιχούν στον κόμβο i με το ελάχιστο s_{ij}, $i \in F^S$.

4.2 ΕΦΑΡΜΟΓΗ ΣΤΟ ΠΡΟΒΛΗΜΑ ΑΝΤΙΣΤΟΙΧΗΣΗΣ

Στη συνέχεια θα περιγράψουμε πως μπορεί ο αλγόριθμος Παπαρρίζου με ξεκίνημα το δάσος ΑΚΡ να εφαρμοστεί για τη επίλυση του προβλήματος αντιστοίχησης. Η παρουσίαση του αλγορίθμου αυτού για το πρόβλημα αντιστοίχησης έγινε στην εργασία **[P1]**. Θα παραθέσουμε την περιγραφή του αλγορίθμου, τη βηματική του εκτέλεση σε μορφή ψευδοκώδικα καθώς και λεπτομέρειες για έναν αποτελεσματικό τρόπο προγραμματισμού του αλγορίθμου.

4.2.1 Περιγραφή του δάσους ΑΚΡ για το πρόβλημα Αντιστοίχησης

Το δάσος ΑΚΡ για το πρόβλημα αντιστοίχησης κατασκευάζεται πολύ εύκολα. Έστω λοιπόν θέλουμε να λύσουμε ένα πρόβλημα αντιστοίχησης για το οποίο μας δίνεται ένας τετραγωνικός πίνακας δεδομένων c διάστασης n. Έστω F το δάσος ΑΚΡ. Το δάσος ΑΚΡ θα αποτελείται από n υπόδεντρα T_j, όπου T_j είναι ένα δέντρο που έχει ρίζα τον κόμβο στήλη j. Τα βασικά δέντρα του δάσους υπολογίζονται ως εξής. Έστω $u_i(F)$, $v_j(F)$, $i, j = 1,...,n$ είναι οι δυϊκές μεταβλητές που αντιστοιχούν στους κόμβους γραμμές και στους κόμβους στήλες του προβλήματος. Έστω τώρα S, D είναι τα σύνολα των κόμβων γραμμών και των κόμβων στηλών του προβλήματος αντίστοιχα. Αρχικά θέτουμε

$$v_j(F) = 0, \quad \forall j \in D$$

Στη συνέχεια υπολογίζουμε τις δυϊκές μεταβλητές των κόμβων γραμμών $u_i(F)$ θέτοντας

$$u_i(F) = \min\left\{c_{ij} : j = 1,...,n\right\}, \quad \forall i \in S$$

Τώρα είναι εύκολο να συμπεράνει κανείς ότι $\forall i \in S$ θα υπάρχει $t \in D$ ώστε $u_i(F) = c_{it}$. Τότε το τόξο (i,t) θα είναι ένα βασικό τόξο του δάσους F. Κάθε βασικό τόξο (i, j) του δάσους F αντιστοιχεί σε μια μεταβλητή απόφασης x_{ij} τέτοια ώστε $x_{ij} = 0$ αν το τόξο κατευθύνεται από πάνω προς τα κάτω και $x_{ij} = 1$ διαφορετικά. Επειδή όμως όλα τα τόξα του δάσους F κατευθύνονται από πάνω προς τα κάτω, θέτουμε $x_{ij} = 1 \; \forall (i, j) \in F$. Τα μειωμένα κόστη s_{ij} μπορούν τώρα εύκολα να υπολογιστούν θέτοντας

$$s_{ij}(F) = c_{ij} - u_i(F) - v_j(F)$$

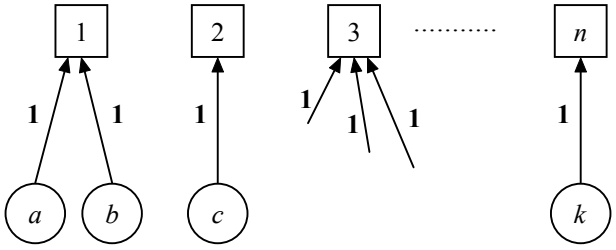
Θα είναι πάντα $s_{ij}(F) \geq 0$, πράγμα που σημαίνει ότι το δάσος ΑΚΡ είναι δυϊκά εφικτό(*dual feasible*). Στη συνέχεια, θα πρέπει να υπολογίσουμε τα σύνολα F^S και F^D που χρησιμεύουν στον προσδιορισμό του εισερχόμενου τόξου (g,h). Για να προσδιορίσουμε τα παραπάνω σύνολα θα πρέπει να ξέρουμε το βαθμό κάθε κόμβου στήλη j. Έστω λοιπόν d_j ο βαθμός του κόμβου στήλη j. Τα σύνολα F^S και F^D προσδιορίζονται με βάση τις παρακάτω σχέσεις.

$$F^S = \{T_j : d_j \geq 2\}$$

ενώ

$$F^D = \{T_j : d_j < 2\}$$

Παρατηρούμε δηλαδή ότι στο σύνολο F^S περιέχονται τα δέντρα εκείνα T_j για τα οποία ο κόμβος j έχει τουλάχιστον 2 παιδιά. Στο σύνολο F^D θα περιέχονται όλα τα υπόλοιπα δέντρα T_j. Παρακάτω, βλέπουμε μια γενική εικόνα του δάσους ΑΚΡ για ένα $n \times n$ πρόβλημα αντιστοίχησης.



Εικόνα 4.10 – Το δάσος ΑΚΡ στη γενική μορφή του

Στη συνέχεια θα παρουσιάσουμε ένα παράδειγμα υπολογισμού του δάσους ΑΚΡ.

<u>Παράδειγμα 4.3</u>

Δίνεται ο πίνακας κόστους

$$C = \begin{bmatrix} 3 & -3 & 19 & 10 & -27 \\ 0 & -1 & 42 & 46 & 77 \\ 70 & 32 & 43 & 71 & 90 \\ 34 & 1 & 0 & 0 & 8 \\ 1 & 7 & 34 & 65 & -7 \end{bmatrix}$$

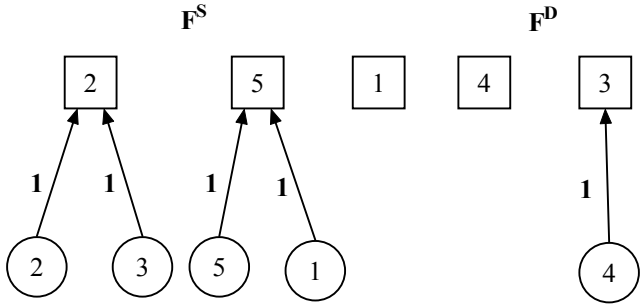
για ένα 5×5 ***πρόβλημα Αντιστοίχησης. Να προσδιορισθεί το δάσος ξεκινήματος ΑΚΡ και να υπολογισθούν τα στοιχεία*** $x_{ij}(F),\ (i,j) \in F$**,** $u_i(F),\ i = 1,...,n$**,** $v_j(F),\ j = 1,...,n$ ***και*** $s_{ij}(F),\ (i,j) \notin F$.

Λύση

Αρχικά θέτουμε $v_j(F)=0,\ j=1,...,5$ και στη συνέχεια υπολογίζουμε τις δυϊκές μεταβλητές των κόμβων γραμμών θέτοντας

$$u_i(F)=\min\left\{c_{ij}: j=1,...,n\right\},\ \ i=1,...,5$$

Εύκολα βρίσκουμε ότι $u_1(F)=-27,\ u_2(F)=-1,\ u_3(F)=32,\ u_4(F)=0,\ u_5(F)=-7$. Συνεπώς τα βασικά τόξα του δάσους F θα είναι τα τόξα (1,5), (2,2), (3,2), (4,3)[4] και (5,5). Το δάσος λοιπόν που προκύπτει είναι το παρακάτω:



Εικόνα 4.11 – Το δάσος ΑΚΡ του παραδείγματός 4.3

Για το παραπάνω δάσος θα είναι $F^S=\{T_2,T_5\}$ και $F^D=\{T_1,T_3,T_4\}$. Παρατηρείστε ότι στο παραπάνω δάσος οι κόμβοι – στήλες 1,4 είναι απομονωμένοι (*isolated*) και ανήκουν στο σύνολο F^S. Τέλος, ο αρχικός πίνακας s_{ij} των μειωμένων κοστών θα είναι ο παρακάτω:

$$s=\begin{bmatrix}30 & 24 & 46 & 37 & 0\\ 1 & 0 & 43 & 47 & 78\\ 38 & 0 & 11 & 39 & 58\\ 34 & 1 & 0 & 0 & 8\\ 8 & 14 & 41 & 72 & 0\end{bmatrix}$$

4.2.2 Περιγραφή του Αλγορίθμου

Επειδή οι αλγόριθμοι που περιγράφουμε δουλεύουν με ριζωμένα δέντρα, χρειάζεται να μετατρέψουμε το δάσος F σε ένα ριζωμένο δέντρο T. Αυτό μπορεί να γίνει πολύ εύκολα προσθέτοντας έναν ουδέτερο (*neutral*) κόμβο 0 και n τεχνητά (*artificial*) τόξα. Ο κόμβος 0 είναι πλέον η ρίζα του δέντρου. Το δέντρο T που προκύπτει συνδέεται με τα σύνολα F^S και F^D με την παρακάτω σχέση:

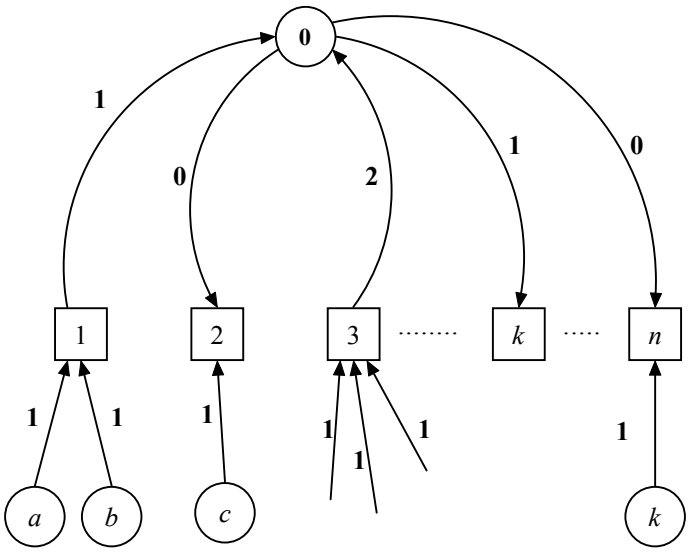
$$T=F\cup\{(r,0),\ \forall T_r\in F^S\}\cup\{(0,w),\ \forall T_w\in F^D\}$$

Παρατηρείστε ότι τα τεχνητά τόξα κατευθύνονται από κόμβους στήλες του συνόλου F^S προς τη ρίζα του δέντρου και από τη ρίζα του δέντρου προς τους κόμβους στήλες του συνόλου F^D. Οι μεταβλητές απόφασης των τεχνητών τόξων που περιέχουν τον κόμβο j - είτε αυτός είναι κόμβος του F^S είτε του F^D - έχουν πάντα θετικές τιμές και ίσες με $\left|d_j-2\right|$

[4] Αντί του τόξου (4,3), βασικό μπορεί να είναι και το τόξο (4,4), αφού $c_{43}=c_{44}=0=\min\left\{c_{4j}\right\},\ j=1,...,5$

(προσέξτε ότι με d_j εδώ συμβολίζουμε το βαθμό του κόμβου στήλη j μετά την εισαγωγή των τεχνητών τόξων).

Συνεπώς το δέντρο που προκύπτει είναι και πρωτευόντως και δυϊκά εφικτό (*primal and dual feasible*). Ένα τέτοιο γενικό δέντρο για ένα $n \times n$ πρόβλημα αντιστοίχησης φαίνεται στην εικόνα 4.12.

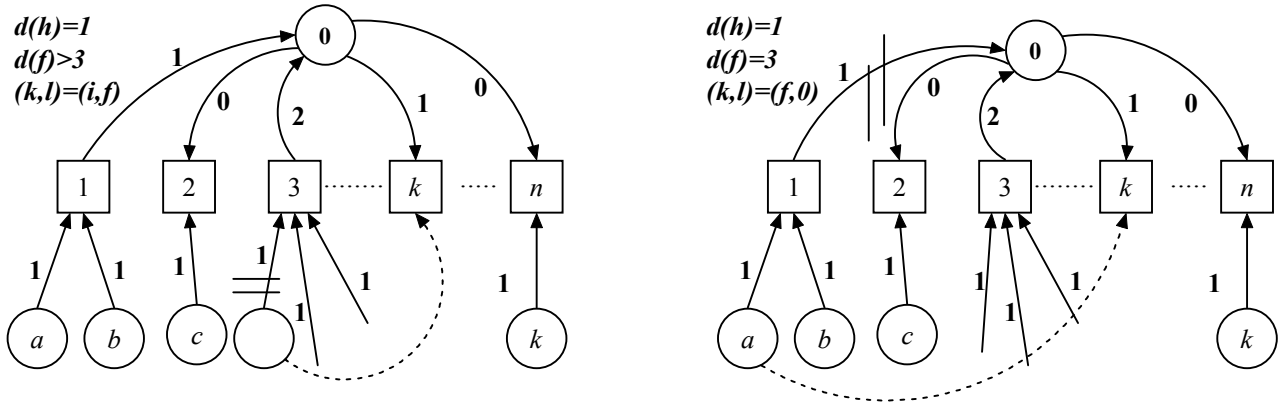


Εικόνα 4.12 – Ένα δέντρο γενικής μορφής που προκύπτει από το δάσος της εικόνας 4.11

Ο αλγόριθμος ξεκινάει με την επιλογή του εισερχόμενου τόξου (g,h). Το εισερχόμενο τόξο επιλέγεται ακριβώς όπως επιλέγεται και στο πρόβλημα Μεταφοράς. Δηλαδή θέτουμε:

$$\delta = s_{gh}(T) = \min\left\{ s_{ij}(T) : i \in F^S,\ j \in F^D \right\}$$

Έτσι, σχηματίζεται ένας διακριτός κύκλος C που αποτελείται πάντα από δύο τεχνητά τόξα. Το εξερχόμενο τόξο (k,ℓ) επιλέγεται με βάση το βαθμό του κόμβου στήλη h. Διακρίνουμε λοιπόν τις εξής περιπτώσεις. Αν $d(h)=2$, τότε εξερχόμενο είναι το μοναδικό προς τα κάτω τόξο (i,h) του δέντρου T. Αν τώρα $d(h)=1 \wedge d(f)>3$, όπου f είναι ο κόμβος του συνόλου F^S που ανήκει στον κύκλο C και είναι προσαρτημένος στη ρίζα του δέντρου T, τότε το εξερχόμενο τόξο είναι το τόξο $(i,f),\ i \neq 0$ που ανήκει στον κύκλο C και είναι προς τα πάνω. Τέλος, αν $d(h)=1 \wedge d(f)=3$, εξερχόμενο θα είναι το τόξο $(f,0)$. Στο σχήμα 4.13 βλέπουμε τις δύο υποπεριπτώσεις επιλογής του εξερχόμενου τόξου όταν $d(h)=1$.



Εικόνα 4.13 – Οι δύο περιπτώσεις επιλογής του εξερχόμενου τόξου όταν *d(h)=1*

Η ανανέωση των μειωμένων κοστών των τόξων του δέντρου γίνεται όπως και στους άλλους αλγορίθμους με βάση τους κόμβους του αποκομμένου δέντρου T^*. Τέλος, οι μεταβλητές απόφασης των τεχνητών τόξων $(j,0)$ ή $(0,j)$ τίθενται πάντα ίσες με $|d_j - 2|$ και τα σύνολα F^S και F^D καθορίζονται πάντα έτσι ώστε $F^S = \{T_j : d_j \geq 3\}$ και $F^D = \{T_j : d_j < 3\}$.

4.2.3 Η Πολυπλοκότητα του Αλγορίθμου

Στη συνέχεια θα προσπαθήσουμε να υπολογίσουμε τη θεωρητική πολυπλοκότητα του αλγορίθμου. Για να το κάνουμε αυτό θα βασιστούμε στη γενική φιλοσοφία του αλγορίθμου. Ο αλγόριθμος μπορεί να χωριστεί σε στάδια(*levels*). Αυτό που γίνεται σε κάθε στάδιο είναι η επανάληψη των περιπτώσεων όπου ισχύει $d(h) = 2$, πράγμα που σημαίνει ότι σε κάθε στάδιο έχουμε διαδοχική αύξηση του συνόλου F^S, αφού κόβονται τόξα $(k,\ell) \in F^D$. Το τέλος ενός σταδίου σηματοδοτείται από την εμφάνιση της περίπτωσης επιλογής εξερχόμενου τόξου όπου ισχύει $d(h) = 1$. Σε αυτήν την περίπτωση, το σύνολο F^S μειώνεται και αυξάνει το σύνολο F^D. Κάθε στάδιο αριθμείται με τον αριθμό των κόμβων-στηλών με βαθμό ίσο με τη μονάδα. Έτσι, το στάδιο k έχει k κόμβους- στήλες w ώστε $d(w) = 1$. Επειδή οι επαναλήψεις ενός σταδίου τερματίζονται όταν εφαρμόζεται η περίπτωση όπου $d(h) = 1$(και κόβεται το μοναδικό προς τα κάτω τόξο (i,h)), εύκολα βγαίνει το συμπέρασμα, ότι αν το στάδιο k έχει k κόμβους στήλες χωρίς παιδία, το στάδιο $k-1$ θα έχει πάντα $k-1$ κόμβους τέτοιου είδους.

Έστω λοιπόν T_k είναι το δέντρο στο στάδιο k και T_{k-1} είναι το δέντρο του αλγορίθμου στο στάδιο $k-1$. Αν F_k^D είναι το σύνολο F^D του σταδίου k, τότε το σύνολο F_k^D μπορεί να περιέχει το πολύ $n-k-1$ κόμβους στήλες που έχουν τουλάχιστον ένα παιδί (με βαθμό τουλάχιστον 2), αφού τουλάχιστον ένας κόμβος στήλη θα περιέχεται στο σύνολο F_k^S . Στη χειρότερη περίπτωση, δηλαδή στην περίπτωση όπου στο στάδιο k, ο πρώτος από τους k κόμβους χωρίς παιδία επιλέγεται ως άκρο του εισερχόμενου τόξου αφού έχουν επιλεγεί όλοι οι άλλοι κόμβοι $j \in F_k^D$ με $d(j) = 2$, συμπεραίνουμε ότι στο στάδιο k θα έχουμε το πολύ $n-k-1$ επαναλήψεις όπου θα εφαρμόζεται η περίπτωση επιλογής εξερχόμενου τόξου με $d(h) = 2$ μέχρι να εφαρμοστεί η περίπτωση όπου $d(h) = 1$. Συνεπώς, μεταβαίνουμε από το δέντρο T_k στο δέντρο T_{k-1} το πολύ σε $n-k$ επαναλήψεις[5].

Για να υπολογίσουμε το συνολικό αριθμό επαναλήψεων του αλγορίθμου, ας θεωρήσουμε ότι αρχικά ο αλγόριθμος ξεκινάει από το στάδιο $n-1$, δηλαδή, στο αρχικό δάσος όλοι οι κόμβοι γραμμές συνδέονται με τον ίδιο κόμβο στήλη t[6] και έτσι υπάρχουν $n-1$ απομονωμένοι κόμβοι. Συνεπώς, για να φτάσει στο στάδιο 0 ο αλγόριθμος θα πρέπει να εκτελέσει το πολύ

$$\sum_{k=1}^{n-1}(n-k) = (n-1) + (n-2) + \ldots + 1 = \frac{n(n-1)}{2}$$

επαναλήψεις, αφού σε κάθε στάδιο k εκτελούνται το πολύ $n-k$ επαναλήψεις και ο αλγόριθμος περνάει το πολύ από $n-1$ στάδια. Αποδείξαμε συνεπώς ότι οι επαναλήψεις που εκτελεί ο αλγόριθμος για την επίλυση ενός προβλήματος Αντιστοίχησης μεγέθους n έχει

---

[5] Ο ακριβής αριθμός των επαναλήψεων εξαρτάται σε ένα βαθμό από τα αρχικά δεδομένα, και τον τρόπο κατασκευής του αρχικού δάσους.

[6] Αυτό μπορεί να συμβεί όταν $\exists\, t \;:\; c_{it} = \min\{c_{ij},\, j = 1,\ldots,n\},\; i = 1,\ldots,n$.

άνω φράγμα $\frac{n(n-1)}{2}$ [7] . Μένει να βρούμε το χρόνο ανά επανάληψη. Όλες οι λειτουργίες του αλγορίθμου(π.χ. ανανεώσεις μεταβλητών) παίρνουν το πολύ χρόνο $O(n)$. Η επιλογή του εισερχόμενου τόξου με βάση τον ορισμό μπορεί να πάρει χρόνο $O(n^2)$, αφού μπορεί να χρειαστούν n^2 συγκρίσεις. Εφαρμόζοντας όμως την τεχνική που χρησιμοποιήθηκε και στο πρόβλημα Μεταφοράς για την επιλογή του εισερχόμενου τόξου (g,h), η πολυπλοκότητα πέφτει στο $O(n)$. Συνεπώς η πολυπλοκότητα $f(n)$ του αλγορίθμου θα είναι

$$f(n) \in \frac{n(n-1)}{2} O(n) = O\left(\frac{n^2(n-1)}{2}\right) = O(n^3)$$ [8]

4.2.4 Βηματική Περιγραφή του Αλγορίθμου

Στη συνέχεια θα παρουσιάσουμε τον αλγόριθμο σε μορφή βημάτων και βασισμένοι στα βήματα του αλγορίθμου θα λύσουμε ένα πρόβλημα.

ΒΗΜΑ 0:*(Καθορισμός αρχικών μεταβλητών)*
Υπολόγισε το δάσος ΑΚΡ με τη μέθοδο που περιγράφηκε. Πρόσθεσε έναν ουδέτερο κόμβο 0 και n τεχνητά τόξα της μορφής $(j,0)$ ή (. Έτσι κατασκευάζεται το αρχικό εφικτό δέντρο T του προβλήματος. Καθόρισε τα σύνολα F^S και F^D έτσι ώστε $F^S = \{T_j : d_j \geq 3\}$ και F^D $\{T :$ $3\}$.

ΒΗΜΑ 1:*(Έλεγχος βελτιστότητας)*

στο βήμα 2.

ΒΗΜΑ 2:*(Επιλογή εισερχομένου τόξου)*
Υπολόγισε το εισερχόμενο τόξο (g,h) ελέγχοντας τα μειωμένα κόστη των μη βασικών τόξων με βάση τον τύπο $\delta = s_{gh} = \min\{s_{ij}( ) :$ $F^S, j \in F^D\}$.

ΒΗΜΑ 3:*(Επιλογή εξερχόμενου τόξου)*

όπου $(i,f$ το μοναδικό προς τα πάνω τόξο που ανήκει στον κύκλο C, αλλιώς αν $d(h) = 1 \wedge$ 3 θέσε $(k,\ell) = (f,0)$.

ΒΗΜΑ 4:*(Ανανέωση των μεταβλητών)*
Υπολόγισε το δέντρο T^* που είναι το δέντρο που αποκόβεται από τη ρίζα όταν αφαιρείται το εξερχόμενο τόξο. Αν $h \in T^*$ θέσε $q =$ δ , αλλιώς θέσε q . Στη συνέχεια εάν ο κόμβος είναι κόμβος-γραμμή θέσε $') = s_{bj}(T) - q$, αλλιώς (δηλαδή ο κόμβος $b \in T$ είναι κόμβος στήλη θέσε $s_{ib}(T$ s $(T) + q$, $i = 1$.(Ανανεώνονται οι τιμές s_{ij} των τόξων) που προσπίπτουν σε κόμβους που ανήκουν στο δέντρο T^*)

[7] Μπορούμε να κατασκευάσουμε δεδομένα –ορισμένα ως συνάρτηση του *n* - που να πετυχαίνουν ακριβώς *n(n – 1)/2* επαναλήψεις.

[8] Θυμίζουμε ότι $f(n) \in O(g(n)) \Leftrightarrow \exists\, c, n_0 : \forall n \geq n_0$ είναι $f(n) \leq cg(n)$

ΒΗΜΑ 5:*(Ανανέωση του τρέχοντος δέντρου)*

Παράδειγμα 4.4
Δίνεται ο πίνακας κόστους

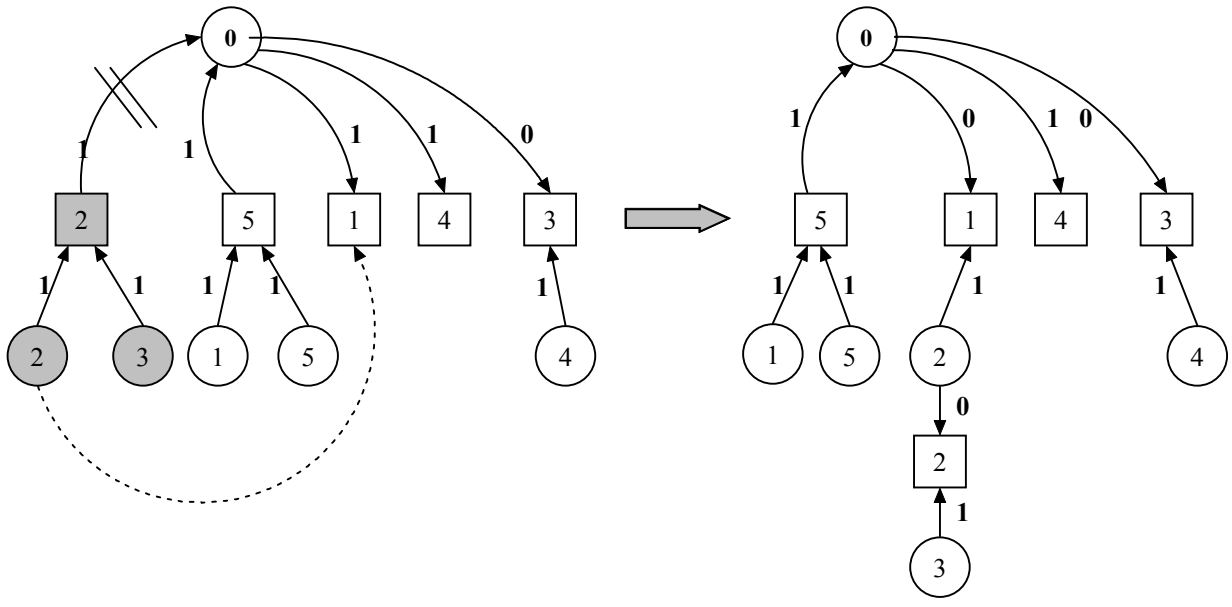
$$C = \begin{bmatrix} 3 & -3 & 19 & 10 & -27 \\ 0 & -1 & 42 & 46 & 77 \\ 70 & 32 & 43 & 71 & 90 \\ 34 & 1 & 0 & 0 & 8 \\ 1 & 7 & 34 & 65 & -7 \end{bmatrix}$$

για ένα 5×5 ***πρόβλημα Αντιστοίχησης. Να λυθεί το πρόβλημα χρησιμοποιώντας τον αλγόριθμο Παπαρρίζου με ξεκίνημα το δάσος ΑΚΡ.***

Λύση

Τα δεδομένα του παραδείγματος είναι ίδια με τα δεδομένα του παραδείγματος 4.3. Συνεπώς, αφού έχουμε υπολογίσει το δάσος ΑΚΡ, προχωράμε στην πρώτη επανάληψη του αλγορίθμου.

Επανάληψη 1



30	24	46	37	**1**	
1	**1**	43	47	78	-1
38	**1**	11	39	58	-1
34	1	**1**	0	8	
8	14	41	72	**1**	
	+1				

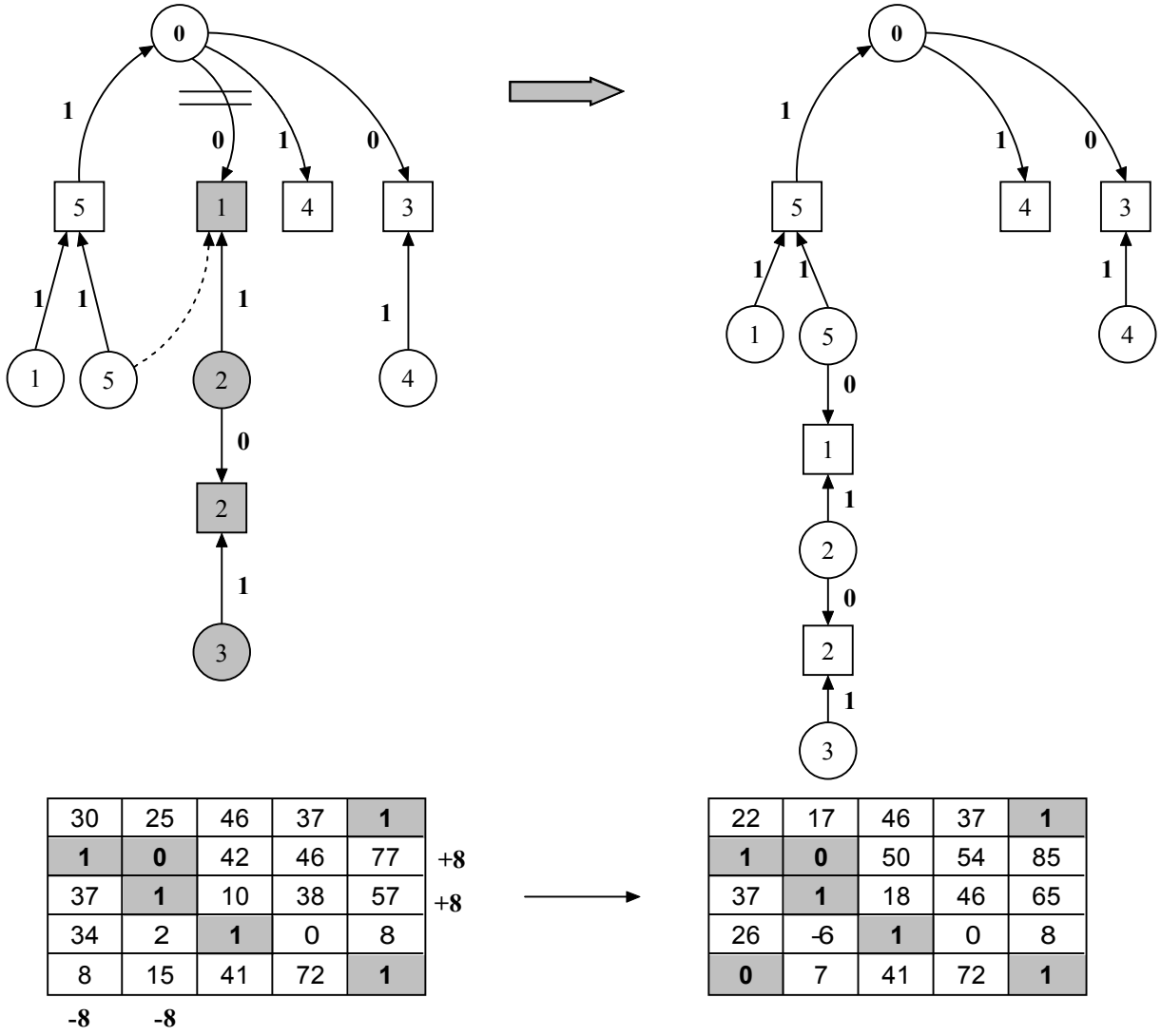
30	25	46	37	**1**
1	**0**	42	46	77
37	**1**	10	38	57
34	2	**1**	0	8
8	15	41	72	**1**

Εικόνα 4.14 – Η πρώτη επανάληψη του αλγορίθμου

- Είναι $F^S = \{1,2,3,5\}$, $F^D = \{1,3,4\}$, $\delta = \min\{s_{ij}(T) : i \in F^S, j \in F^D\} = s_{21} = 1$, συνεπώς $(g,h) = (2,1)$

- Σύμφωνα με όσα αναφέρθηκαν στην περιγραφή του αλγορίθμου θα πρέπει να βρούμε το βαθμό του κόμβου h. Είναι $d(h) = d(1) = 1$, άρα θα πρέπει να ελέγξουμε και το βαθμό του κόμβου f, όπου f είναι ο μοναδικός κόμβος - στήλη που κρέμεται από τη ρίζα του δέντρου και ανήκει στον κύκλο C της επανάληψης. Συνεπώς $f = 2$ και $d(f) = 3$, άρα εξερχόμενο θα είναι το τόξο $(k, \ell) = (f, 0) = (2, 0)$.
- Η ανανέωση των τιμών των μεταβλητών απόφασης και των μειωμένων κοστών γίνεται με βάση τον κύκλο C και του αποκομμένου δέντρου $T*$ και φαίνεται αναλυτικά στο σχήμα 4.14.

Επανάληψη 2



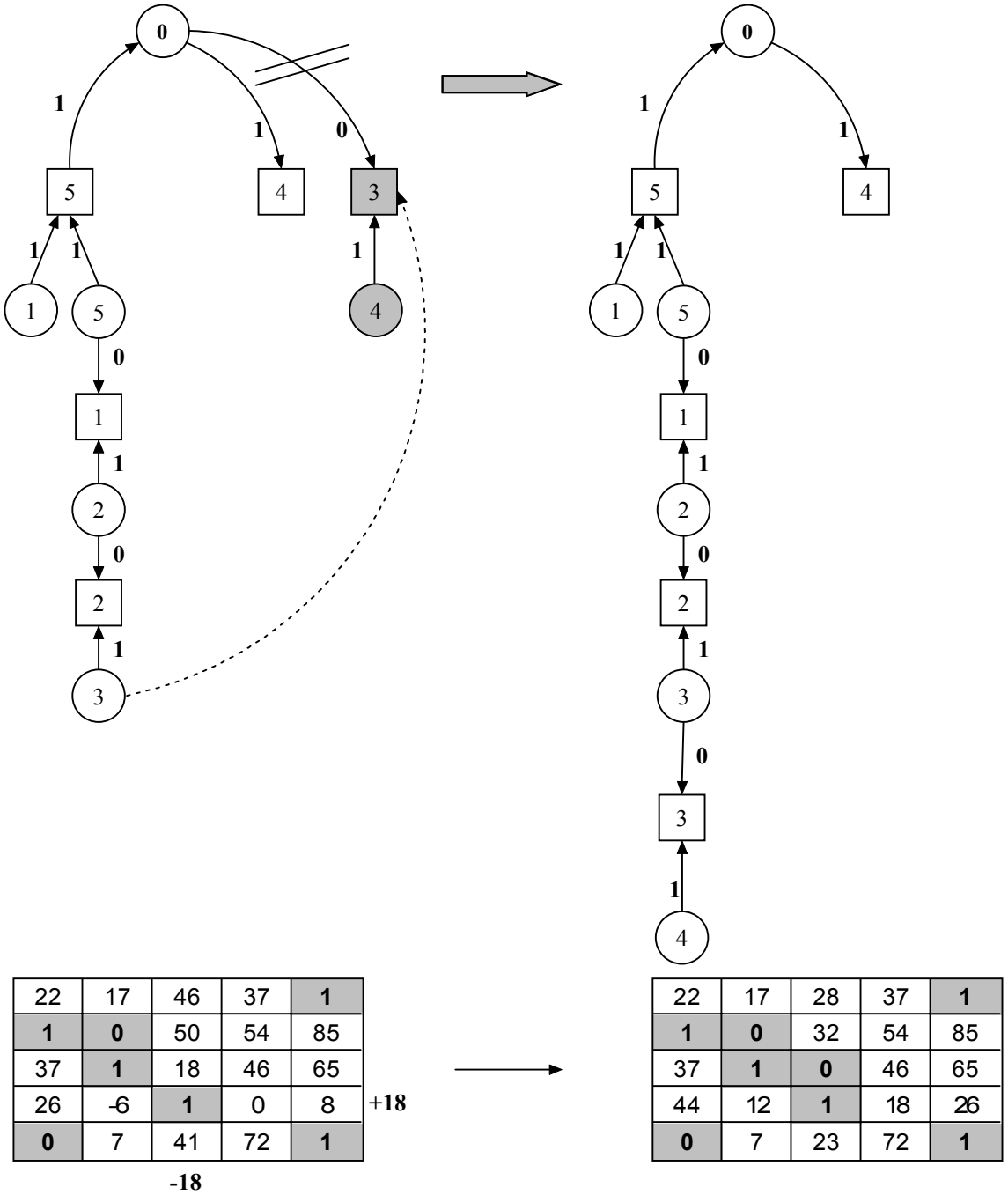
30	25	46	37	**1**	
1	**0**	42	46	77	**+8**
37	**1**	10	38	57	**+8**
34	2	**1**	0	8	
8	15	41	72	**1**	
-8	**-8**				

22	17	46	37	**1**
1	**0**	50	54	85
37	**1**	18	46	65
26	-6	**1**	0	8
0	7	41	72	**1**

Εικόνα 4.15 – Η δεύτερη επανάληψη του αλγορίθμου

- Είναι $F^S = \{1,5\}$, $F^D = \{1,2,3,4\}$, $\delta = \min\{s_{ij}(T) : i \in F^S, j \in F^D\} = s_{51} = 8$, συνεπώς $(g, h) = (5,1)$
- Σύμφωνα με όσα αναφέρθηκαν στην περιγραφή του αλγορίθμου θα πρέπει να βρούμε το βαθμό του κόμβου h. Είναι $d(h) = d(1) = 2$, άρα $(k, \ell) = (p(h), h) = (0,1)$.
- Η ανανέωση των τιμών των μεταβλητών απόφασης και των μειωμένων κοστών γίνεται με βάση τον κύκλο C και του αποκομμένου δέντρου $T*$ και φαίνεται αναλυτικά στο σχήμα 4.15.

Επανάληψη 3



22	17	46	37	**1**
1	**0**	50	54	85
37	**1**	18	46	65
26	-6	**1**	0	8
0	7	41	72	**1**

+18

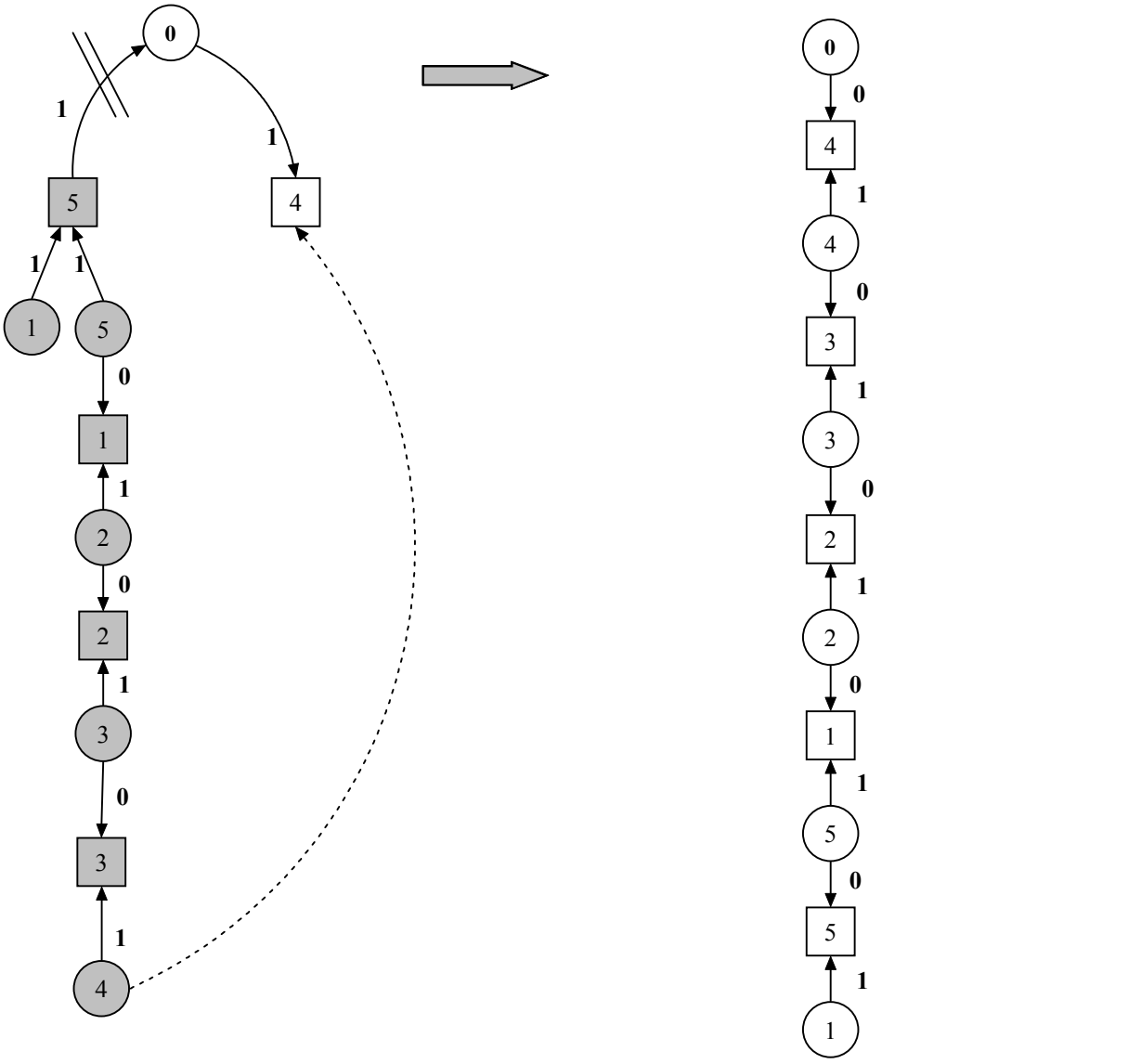
-18

22	17	28	37	**1**
1	**0**	32	54	85
37	**1**	**0**	46	65
44	12	**1**	18	26
0	7	23	72	**1**

Εικόνα 4.16 – Η τρίτη επανάληψη του αλγορίθμου.

- Είναι $F^S = \{1,2,3,5\}$, $F^D = \{3,4\}$, $\delta = \min\{s_{ij}(T) : i \in F^S, j \in F^D\} = s_{33} = 18$, άρα $(g,h) = (3,3)$.
- Σύμφωνα με όσα αναφέρθηκαν στην περιγραφή του αλγορίθμου θα πρέπει να βρούμε το βαθμό του κόμβου h. Είναι $d(h) = d(3) = 2$, άρα $(k,\ell) = (p(h),h) = (0,3)$.
- Η ανανέωση των τιμών των μεταβλητών απόφασης και των μειωμένων κοστών γίνεται με βάση τον κύκλο C και του αποκομμένου δέντρου $T*$ και φαίνεται αναλυτικά στο σχήμα 4.16.
- Επειδή $F^S \neq \varnothing$, ο αλγόριθμος συνεχίζει να εκτελείται.

Επανάληψη 4



22	17	28	37	**1**	**-18**
1	**0**	32	54	85	**-18**
37	**1**	**0**	46	65	**-18**
44	12	**1**	18	26	**-18**
0	7	23	72	**1**	**-18**
+18	**+18**	**+18**		**+18**	

22	17	28	19	**1**
0	**1**	32	36	85
37	**0**	**1**	28	65
44	12	**0**	**1**	26
1	7	23	54	**0**

Εικόνα 4.17 – Η τέταρτη επανάληψη του αλγορίθμου

- Είναι $F^S = \{1,2,3,4,5\}$, $F^D = \{4\}$, $\delta = \min\{s_{ij}(T) : i \in F^S, j \in F^D\} = s_{44} = 18$,συνεπώς $(g,h) = (4,4)$
- Σύμφωνα με όσα αναφέρθηκαν στην περιγραφή του αλγορίθμου θα πρέπει να βρούμε το βαθμό του κόμβου h. Είναι $d(h) = d(4) = 1$, άρα θα πρέπει να ελέγξουμε και το βαθμό του κόμβου f, όπου f είναι ο μοναδικός κόμβος - στήλη που κρέμεται από τη ρίζα του δέντρου και ανήκει στον κύκλο C της επανάληψης. Συνεπώς $f = 5$ και $d(f) = 3$, άρα εξερχόμενο θα είναι το τόξο $(k,\ell) = (f,0) = (5,0)$. Σε αυτήν την

επανάληψη ο αλγόριθμος περνάει στο στάδιο 0 και συνεπώς τερματίζεται η εκτέλεση. Παρατηρείστε ότι είναι $F^S = \varnothing$.

- Η ανανέωση των τιμών των μεταβλητών απόφασης και των μειωμένων κοστών γίνεται με βάση τον κύκλο C και του αποκομμένου δέντρου $T*$ και φαίνεται αναλυτικά στο σχήμα 4.17.

4.2.5 Προγραμματισμός του Αλγορίθμου

Στη συνέχεια θα παρουσιάσουμε τον ψευδοκώδικα του αλγορίθμου και τις πιο σημαντικές συναρτήσεις.

Αλγόριθμος Α3
Ο παρακάτω αλγόριθμος λύνει το πρόβλημα Αντιστοίχησης , χρησιμοποιώντας το αλγόριθμο Παπαρρίζου με ξεκίνημα το δάσος AKP.
Είσοδος: *Ο πίνακας κόστους* **C(nxn)**.
Έξοδος : *Η λύση του προβλήματος Αντιστοίχησης* **X(nxn)**.

```
1.  [X,S]=AKPtree(C,n);
2.  p=NodeFather(X,n);
3.  d=InitializeNodeDepth(p,n);
4.  [Fs,Fd]=SupplyDemandSets(X,n,p);
5.  Ematrix=Initialize(S,Fs,Fd,n);
6.  Fs=[];
7.  Fd=[];
8.  while count<2*n
9.    [Fs,Fd]=SupplyDemandSets(X,n,p);
10.   count=0;
11.   for q=1:2*n
12.     if Fs(q)==0
13.       count=count+1;
14.     end
15.   end
16.   if count<2*n
17.     iterations=iterations+1;
18.     [min,g,h,f,Circle]=EnteringArc(S,Fs,Fd,n,p,Ematrix);
19.     xv=ArcHelp(Fs,Fd,p,n);
20.     deg=degree(X,xv,n);
21.     [k,l,xv,p,upwards,Tcut]=LeavingArc(g,h,deg,f,Circle,p,n,xv);
22.     [e1,e2,f1,z]=Steam(p,g,h,k,l,d,n,Tcut);
23.     d=UpdateNodeDepth(p,n,Tcut,z,d,e1,e2,f1);
24.     p=UpdateNodeFather(p,z,e2,f1);
25.     [Ematrix,S,X]=UpdateVariables(....[9]);
26.   end
27. end
28. [Fs,Fd]=SupplyDemandSets(X,n,p);
```

Αλγόριθμος 4.2.1 – Ο κώδικας υλοποίησης του αλγορίθμου

Παρατηρούμε ότι ο παραπάνω κώδικας δεν διαφέρει σημαντικά στη φιλοσοφία του από τους κώδικες των άλλων αλγορίθμων. Στις πρώτες γραμμές γίνεται η αρχικοποίηση των μεταβλητών που θα χρησιμοποιήσει ο αλγόριθμος. Έτσι, αρχικοποιούνται το δάσος AKP, το διάνυσμα βάθους, το διάνυσμα πατέρα-κόμβου, ο πίνακας **Ematrix** που χρησιμεύει στην επιλογή του εισερχόμενου τόξου (g,h).

- **[k,l,xv,p,upwards,Tcut]=LeavingArc(g,h,deg,f,Circle,p,n,xv)**

[9] **g,h,k,l,d,upwards,n,p,X,S,Tcut,min,Ematrix,deg,Fs,Fd**

Αλγόριθμος LeavingArc
Ο παρακάτω αλγόριθμος καθορίζει το εξερχόμενο τόξο .
Είσοδος: g, h, deg(n), f, Circle(), p(2n), n, xv()
Έξοδος : k, l, xv(), p(2n), upwards, Tcut

```
1.  function [k,l,xv,p,upwards,Tcut]=LeavingArc(g,h,deg,f,Circle,p,n,xv)
2.  if deg(h)==2
3.    k=p(h+n);
4.    l=h;
5.    start=l+n;
6.    Tcut=Preorder(start,p,n);
7.    if p(h+n)==0
8.      xv(h)=inf;
9.    end
10.   upwards=0;
11. end
12. if deg(h)==1
13.   if deg(f-n)>3
14.     upwards=1;
15.     i=1;
16.     while Circle(i)~=0
17.       if p(Circle(i))==f
18.         k=Circle(i);
19.         l=f-n;
20.       end
21.     i=i+1;
22.   end
23.   start=k;
24.   Tcut=Preorder(start,p,n);
25.   if p(h+n)==0
26.     xv(h)=0;
27.   else
28.    i=i+1;
29.    while Circle(i)~=0
30.      i=i+1;
31.    end
32.   xv(Circle(i)-n)=0;
33.   end
34.   xv(f-n)=xv(f-n)-1;
35.   elseif deg(f-n)==3
36.     k=f-n;
37.     l=0;
38.     start=f;
39.     Tcut=Preorder(start,p,n);
40.     xv(f-n)=inf;
41.     upwards=1;
42.     i=1;
43.     while Circle(i)~=h+n
44.       i=i+1;
45.     end
46.     if p(h+n)==0
47.       xv(h)=0;
48.     else
49.      j=i;
50.      while Circle(j)~=0
51.        j=j-1;
52.      end
53.     xv(Circle(j)-n)=0;
54.     end
55.   end
56. end
```

Αλγόριθμος 4.2.2 – Επιλογή του εξερχόμενου τόξου

Στον παραπάνω κώδικα βλέπουμε ότι επιλογή του εξερχόμενου τόξου γίνεται ακριβώς όπως περιγράφηκε στη θεωρία, δηλαδή ανάλογα με το βαθμό του κόμβου του κόμβου h.

Συνεπώς, ελέγχονται όλες οι περιπτώσεις που αναφέρθηκαν στην περιγραφή του αλγορίθμου ($(d(h)=2)\vee(d(h)=1\wedge d(f)>3)\vee(d(h)=1\wedge d(f)=3)$). Ο αναλυτικός έλεγχος γίνεται για την πρώτη περίπτωση στις γραμμές 2-11, ενώ στις υπόλοιπες γραμμές εξετάζονται οι άλλες περιπτώσεις.

- **xv = ArcHelp(Fs,Fd,p,n)**

Αλγόριθμος ArcHelp
Αποθήκευση των μεταβλητών απόφασης των τεχνητών τόξων .
Είσοδος: Fs(), Fd(), p(2n), n
Έξοδος : xv(n)

```
1.  function xv=ArcHelp(Fs,Fd,p,n)
2.  i=1;
3.  while Fs(i)~=0
4.    if ((Fs(i)>n)&(p(Fs(i))==0))
5.    degnow1=0;
6.    for j=1:n
7.      if p(j)==Fs(i)
8.        degnow1=degnow1+1;
9.      end
10.   end
11.   xv(Fs(i)-n)=degnow1-1;
12.   elseif (Fs(i)>n)&(p(Fs(i))~=0)
13.     xv(Fs(i)-n)=inf;
14.   end
15.   i=i+1;
16. end
17. j=1;
18. while Fd(j)~=0
19.   if (Fd(j)>n)&(p(Fd(j))==0)
20.     degnow2=0;
21.     for k=1:n
22.       if p(k)==Fd(j)
23.         degnow2=degnow2+1;
24.       end
25.     end
26.     if degnow2==0
27.       xv(Fd(j)-n)=-1;
28.     else
29.       xv(Fd(j)-n)=0;
30.     end
31.   elseif (Fd(j)>n)&(p(Fd(j))~=0)
32.     xv(Fd(j)-n)=inf;
33.   end
34. j=j+1;
35. end
```

Αλγόριθμος 4.2.3 – Αποθήκευση των μεταβλητών απόφασης των τεχνητών τόξων

Με την παραπάνω συνάρτηση, παράγουμε ένα διάνυσμα **xv** n θέσεων όπου μπορούμε να αποθηκεύσουμε τις μεταβλητές απόφασης των τεχνητών τόξων του δέντρου. Οι τιμές που παίρνει το διάνυσμα κωδικοποιούνται ως εξής:

- Αν $xv(j)>0$, τότε έχουμε ένα τεχνητό τόξο της μορφής $(j,0)$ μέσω του οποίου μεταφέρονται $xv(j)$ μονάδες, δηλαδή ισχύει $x_{j0}=xv(j)$.
- Αν $xv(j)\le 0$, τότε έχουμε ένα τεχνητό τόξο της μορφής $(0,j)$ μέσω του οποίου μεταφέρονται $-xv(j)$ μονάδες, δηλαδή ισχύει $x_{0j}=-xv(j)$.
- Αν $xv(j)=\infty$, τότε δεν υπάρχει τεχνητό τόξο της μορφής που να περιέχει τον κόμβο j.

4.3ΑΝΑΦΟΡΕΣ

[P1] H. Achatz, P. Kleinschmidt and K. Paparrizos, "*A dual forest algorithm for the assignment problem,*" Dimacs, vol. 4, pp. 1-10, 1990.

[P2] K. Paparrizos, *"A non improving simplex algorithm for transportation problems"*, Operations Research, vol 30, pp 1-15, 1996

[P3] K. Paparrizos, *"An Infeasible (Exterior Point) Simplex Algorithm for Assignment Problems"*, Mathematical Programming 51 (1991) 45-54 North Holland

[PPS] C. Papamanthou, K. Paparrizos, N. Samaras, *"A Comparative Computational Study of Exterior Point Algorithms for Dense Assignment Problems"* in preparation

[Cun] Cunningham, W.H. (1976),"*A Network Simplex Method"*, Math Prog 11,105-116

[CLR] Cormen, Leiserson, Rivest, *"Introduction to Algorithms"*, *MIT Press*, (1990)

[JB] Jensen and Barnes, (1980),*"Network Flow Programming"*, Malabar, Fla.: R.E. Krieger Pub. Co., 1980

[Pap1] K. Paparrizos, *"Linear Programming – Algorithms and Applications"*, in Greek

[Ber] Bertsekas D.**, "***Network Optimization: Continuous and Discrete Models"*, Athena Scientific, 1998

[Kn] D.E. Knuth, *"The Art of Computer Programming – Fundamental Algorithms"*, 3rd edition, Addison-Wesley, 1997

[Roc] Rockafellar R.T., **"***Network Flows and Monotropic Optimization»*, Wiley-Interscience, 1984 (610 pp.).

[Pap2] K. Paparrizos, *"Network Programming"*, Lecture notes in Greek

[Pap3] K. Paparrizos, *"Algorithms – Analysis, Design and Complexity"*, Lecture notes in Greek

[SS] Stephanides G., Samaras N., *"Computational Methods with Matlab"*, Zygos Publications, Thessaloniki 1999, in Greek

[PK] Paparrizos K., *"Matlab 6"*, Zygos Publications, Thessaloniki 2001, in Greek

ΚΕΦΑΛΑΙΟ 5

Ο ΑΛΓΟΡΙΘΜΟΣ ΠΑΠΑΡΡΙΖΟΥ

(ΞΕΚΙΝΗΜΑ ΜΕ ΔΑΣΟΣ ΑΠΛΟΥ ΞΕΚΙΝΗΜΑΤΟΣ)

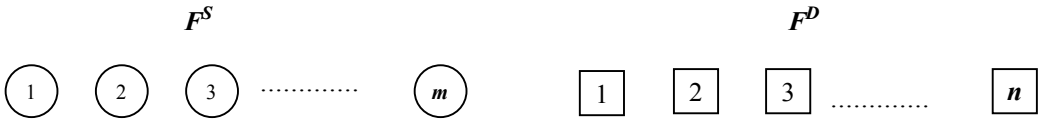
Στο κεφάλαιο αυτό θα παρουσιάσουμε τον αλγόριθμο Παπαρρίζου με ξεκίνημα το δάσος απλού ξεκινήματος. Ο αλγόριθμος αυτός παρουσιάσθηκε πρώτη φορά για το πρόβλημα Μεταφοράς στην εργασία**[P1]** από τον καθηγητή του τμήματος Κ. Παπαρρίζο. Πρόκειται για έναν αλγόριθμο τύπου Simplex(*Simplex Type Algorithm*). Το δάσος απλού ξεκινήματος που χρησιμοποιεί ο αλγόριθμος δεν είναι ούτε πρωτίστως ούτε δϋικά εφικτό.

5.1 ΕΦΑΡΜΟΓΗ ΣΤΟ ΠΡΟΒΛΗΜΑ ΜΕΤΑΦΟΡΑΣ

Ο αλγόριθμος λύνει ένα $m \times n$ πρόβλημα Μεταφοράς σε χρόνο το πολύ $O\big(d(m+n)D\big)$ και το πολύ σε $dD - d(d-1)/2$ επαναλήψεις , όπου D η συνολική ζήτηση και $d = \min\{m, n\}$ **[P1]**. Στη συνέχεια θα παρουσιάσουμε την αρχική λύση που χρησιμοποιεί ο αλγόριθμος που θα περιγράψουμε.

5.1.1 Περιγραφή του Δάσους Απλού Ξεκινήματος για το πρόβλημα Μεταφοράς

Το δάσος απλού ξεκινήματος είναι πάρα πολύ εύκολο στην κατασκευή του. Το δάσος απλού ξεκινήματος δεν περιέχει καθόλου τόξα αλλά αποτελείται μόνο από κόμβους. Επίσης, όπως και στο δάσος ΑΚΡ, το δάσος απλού ξεκινήματος χωρίζεται σε δύο σύνολα το πλεονασματικό δάσος F^S (*surplus forest*) και το ελλειμματικό δάσος F^D (*deficit forest*). Το δάσος F^S περιέχει αρχικά όλους τους κόμβους γραμμές, ενώ το δάσος F^D περιέχει αρχικά όλους τους κόμβους στήλες. Στη γενικότερη περίπτωση, μπορούμε να πούμε ότι το δάσος απλού ξεκινήματος αποτελείται από $m+n$ υπόδεντρα T_w, όπου w μπορεί να είναι ή κόμβος γραμμή ή κόμβος στήλη. Ένα γενικό δάσος απλού ξεκινήματος για ένα $m \times n$ πρόβλημα Μεταφοράς φαίνεται στο παρακάτω σχήμα:



Εικόνα 5.0 – Ένα γενικό δάσος απλού ξεκινήματος για ένα πρόβλημα Μεταφοράς

Αρχικά, οι δυϊκές μεταβλητές όλων των κόμβων του δάσους τίθενται ίσες με το μηδέν. Έτσι θα είναι

$$u_i(F) = 0, \ \forall i \in F^S$$

και

$$v_j(F) = 0, \ \forall j \in F^D$$

Συνεπώς, τα μειωμένα κόστη των τόξων (i, j) θα είναι $s_{ij}(F) = c_{ij} - u_i(F) - v_j(F) = c_{ij}$

<u>Παράδειγμα 5.1</u>

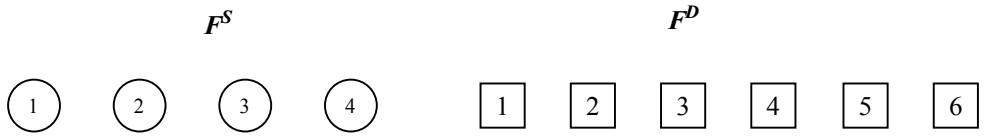
Δίνεται ο πίνακας κόστους

$$C = \begin{bmatrix} -5 & 49 & 28 & 20 & 47 & 43 \\ 19 & 62 & 32 & 61 & 0 & 13 \\ -2 & 63 & 14 & 46 & 28 & 53 \\ 1 & 50 & 41 & 26 & 59 & 0 \end{bmatrix}$$

τα διανύσματα προσφοράς και ζήτησης $a=\begin{bmatrix}5 & 2 & 7 & 6\end{bmatrix}$ ***και*** $b=\begin{bmatrix}2 & 3 & 7 & 2 & 1 & 5\end{bmatrix}$ ***αντίστοιχα για ένα*** 4×6 ***πρόβλημα Μεταφοράς. Να προσδιορισθεί το δάσος απλού ξεκινήματος και να υπολογισθούν τα στοιχεία,*** $u_i(F),\ i=1,...,m$ **,** $v_j(F),\ j=1,...,n$ ***και*** $s_{ij}(F),\ (i,j)\notin F$ **.**

Λύση:

Το δάσος απλού ξεκινήματος για το 4×6 πρόβλημα Μεταφοράς που έχουμε να λύσουμε είναι το παρακάτω:



Εικόνα 5.1 – Το δάσος απλού ξεκινήματος του παραδείγματος

Τα στοιχεία που αντιστοιχούν στο παραπάνω δάσος θα είναι $u=\begin{bmatrix}0 & 0 & 0 & 0\end{bmatrix}$, $v=\begin{bmatrix}0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}$ και

$$S=C=\begin{bmatrix}-5 & 49 & 28 & 20 & 47 & 43\\ 19 & 62 & 32 & 61 & 0 & 13\\ -2 & 63 & 14 & 46 & 28 & 53\\ 1 & 50 & 41 & 26 & 59 & 0\end{bmatrix}.$$

5.1.2 Περιγραφή του Αλγορίθμου

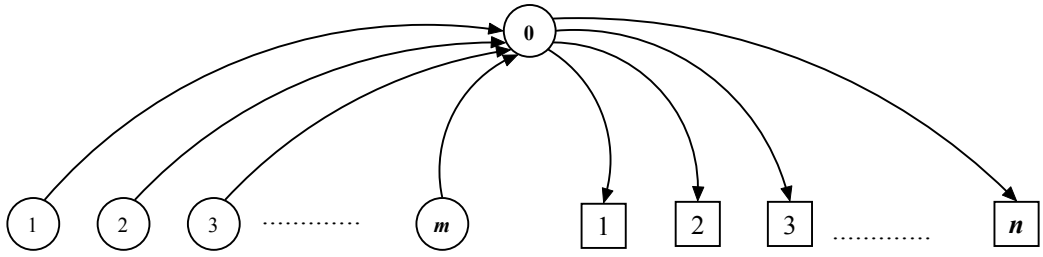
Όπως και οι προηγούμενοι αλγόριθμοι που παρουσιάσθηκαν, έτσι και αυτός ο αλγόριθμος δουλεύει με δέντρα. Συνεπώς, πρέπει να βρούμε έναν τρόπο μετατροπής του δάσους απλού ξεκινήματος σε ένα ριζωμένο δέντρο. Αυτό μπορεί εύκολα να επιτευχθεί αν προσθέσουμε $m+n$ τεχνητά τόξα (*artificial arcs*) και έναν ουδέτερο κόμβο (*neutral node*) 0. Τα τεχνητά τόξα του δέντρου θα είναι της μορφής $(i,0)$ για κάθε κόμβο γραμμή i και αντίστοιχα θα είναι της μορφής $(0,j)$ για κάθε κόμβο στήλη j. Αρχικά υπολογίζουμε τις μεταβλητές απόφασης των τεχνητών τόξων $(i,0)$ και $(0,j)$ με τρόπο που να ικανοποιούνται οι περιορισμοί του προβλήματος. Συνεπώς, θέτουμε

$$x_{i0}=a_i\geq 0$$

και

$$x_{0j}=b_j\geq 0$$

, όπου a το m - διάστατο διάνυσμα προσφοράς και b το n - διάστατο διάνυσμα ζήτησης.



Εικόνα 5.2- Το δέντρο που παράγεται από το δάσος της εικόνας 5.0

Ένα γενικό δέντρο που παράγεται από ένα δάσος απλού ξεκινήματος είναι αυτό του σχήματος 5.2. Κατά τη διάρκεια εκτέλεσης του αλγορίθμου τα σύνολα F^S και F^D προσδιορίζονται έτσι ώστε

$$F^S = \{T_i : i \in I\}$$

και

$$F^D = \{T_j : j \in J\}$$

,όπου I το σύνολο των κόμβων γραμμών του προβλήματος και J το σύνολο κόμβων στηλών του προβλήματος.

Ο αλγόριθμος έχει την ίδια περίπου φιλοσοφία με αυτόν που παρουσιάσθηκε στο προηγούμενο κεφάλαιο. Αρχικά, επιλέγεται το εισερχόμενο τόξο (g,h) χρησιμοποιώντας τη σχέση

$$\delta = s_{gh} = \min\{s_{ij}(T) : i \in F^S, j \in F^D\}$$

Το εξερχόμενο τόξο (k,ℓ) επιλέγεται με τον κανόνα του Cunninghum**[Cun]** με τον τρόπο που είδαμε και στους προηγούμενους αλγορίθμους. Οι ανανεώσεις των μεταβλητών απόφασης καθώς και των μειωμένων κοστών των τόξων (i,j) γίνεται ακριβώς με τον ίδιο τρόπο που είδαμε και στους άλλους αλγορίθμους. Ακολουθεί η βηματική περιγραφή του αλγορίθμου.

5.1.3 Βηματική Περιγραφή του Αλγορίθμου

Στη συνέχεια θα παρουσιάσουμε τον αλγόριθμο σε μορφή βημάτων και βασισμένοι στα βήματα του αλγορίθμου θα λύσουμε ένα πρόβλημα.

ΒΗΜΑ 0:*(Καθορισμός αρχικών μεταβλητών)*
Υπολόγισε το δάσος απλού ξεκινήματος F. Καθόρισε τα σύνολα F^S και F^D με τη μέθοδο που περιγράψαμε. Κατασκεύασε το αρχικό δέντρο T προσθέτοντας έναν τεχνητό κόμβο 0 και τα τεχνητά τόξα $(i,0)$, θέτοντας $x_{i0} = a_i \geq 0$ για κάθε $i \in F^S$ και $(0,j)$, θέτοντας $x_{0j} = b_j \geq 0$ για κάθε $j \in F^D$.

ΒΗΜΑ 1:*(Έλεγχος βελτιστότητας)*

κόμβος γραμμή, τότε το δέντρο T είναι βέλτιστο και ο αλγόριθμος τερματίζει, αλλιώς πήγαινε στο βήμα 2.

ΒΗΜΑ 2:*(Επιλογή εισερχομένου τόξου)*
Υπολόγισε το εισερχόμενο τόξο (g,h) ελέγχοντας τα μειωμένα κόστη των μη βασικών τόξων με βάση τον τύπο $\delta = s_{gh} = \min\{s_{ij}(T) : i \in F^S, j \in F^D\}$.

ΒΗΜΑ 3:*(Επιλογή εξερχόμενου τόξου)*
Υπολόγισε τα σύνολα τόξων C^+, C^- και καθόρισε τη μεταβλητή ε, θέτοντας $\varepsilon = \min\{x_{ij}(T) : (i,j) \in C^-\}$.Βρες το σύνολο C' των επιλέξιμων τόξων, θέτοντας $C' = \{(i,j) \in C^- : \ x_{ij}(T) = \varepsilon\}$. Επέλεξε το εξερχόμενο τόξο $(k,\ell) \in C^-$ με τον κανόνα του

Cunningham, δηλαδή το εξερχόμενο τόξο θα είναι το πρώτο επιλέξιμο τόξο $(r,t) \in C^-$ που συναντάμε κατά την διάσχιση του κύκλου C, ξεκινώντας από τη ρίζα του δέντρου.

ΒΗΜΑ 4:*(Ανανέωση των μεταβλητών)*
Θέσε $x_{ij}(T') = x_{ij}(T) - \varepsilon$ για κάθε τόξο $) \in C^-$ και αντιστοίχως $x_{ij}(T')$ + για κάθε τόξο $(i,j) \in C^+$. Στη συνέχεια ανανέωσε τα σύνολα F^S, F^D. Υπολόγισε το δέντρο T^* που είναι το δέντρο που αποκόβεται από τη ρίζα όταν αφαιρείται το εξερχόμενο τόξο. Αν $h \in T^*$ θέσε $q = -\delta > 0$, αλλιώς θέσε q < 0. Στη συνέχεια εάν ο κόμβος είναι κόμβος-γραμμή θέσε $s_{bj}(T') = s_{bj}(T) - q$, $= 1,...,n$ αλλιώς (δηλαδή ο κόμβος είναι κόμβος-στήλη) θέσε $s_{ib}(T') = s_{ib}(T) + q$ $= 1,...,m$.(Ανανεώνονται οι τιμές των τόξων (i,j) που προσπίπτουν σε κόμβους που ανήκουν στο δέντρο T^*)

ΒΗΜΑ 5:*(Ανανέωση του τρέχοντος δέντρου)*

Μια πιο αυστηρή βηματική περιγραφή του αλγορίθμου περιέχεται στην εργασία **[P1]**.

Στη συνέχεια θα παρουσιάσουμε την ακριβή επίλυση ενός παραδείγματος. Θα χρησιμοποιήσουμε τα δεδομένα που χρησιμοποιήσαμε στο παράδειγμα 5.1.

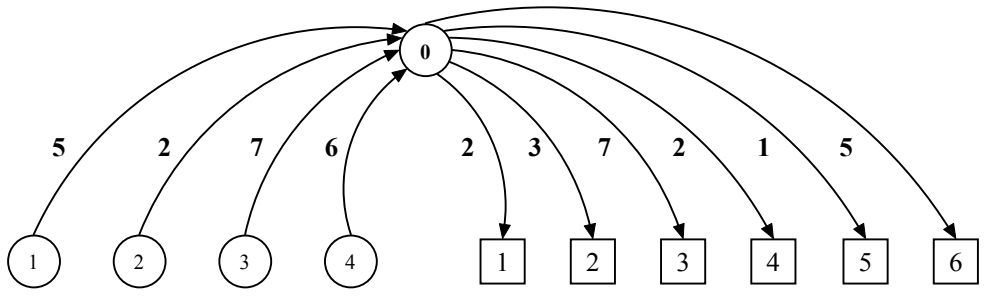
Παράδειγμα 5.2

Δίνεται ο πίνακας κόστους

$$C = \begin{bmatrix} -5 & 49 & 28 & 20 & 47 & 43 \\ 19 & 62 & 32 & 61 & 0 & 13 \\ -2 & 63 & 14 & 46 & 28 & 53 \\ 1 & 50 & 41 & 26 & 59 & 0 \end{bmatrix}$$

τα διανύσματα προσφοράς και ζήτησης $a = \begin{bmatrix} 5 & 2 & 7 & 6 \end{bmatrix}$ ***και*** $b = \begin{bmatrix} 2 & 3 & 7 & 2 & 1 & 5 \end{bmatrix}$ ***αντίστοιχα για ένα*** 4×6 ***πρόβλημα Μεταφοράς. Να λυθεί με τον αλγόριθμο Παπαρρίζου και να χρησιμοποιηθεί ως αρχική εφικτή λύση το δάσος απλού ξεκινήματος.***

Λύση:
Το δέντρο ξεκινήματος που προκύπτει από το δάσος του παραδείγματος 5.1 είναι το παρακάτω:

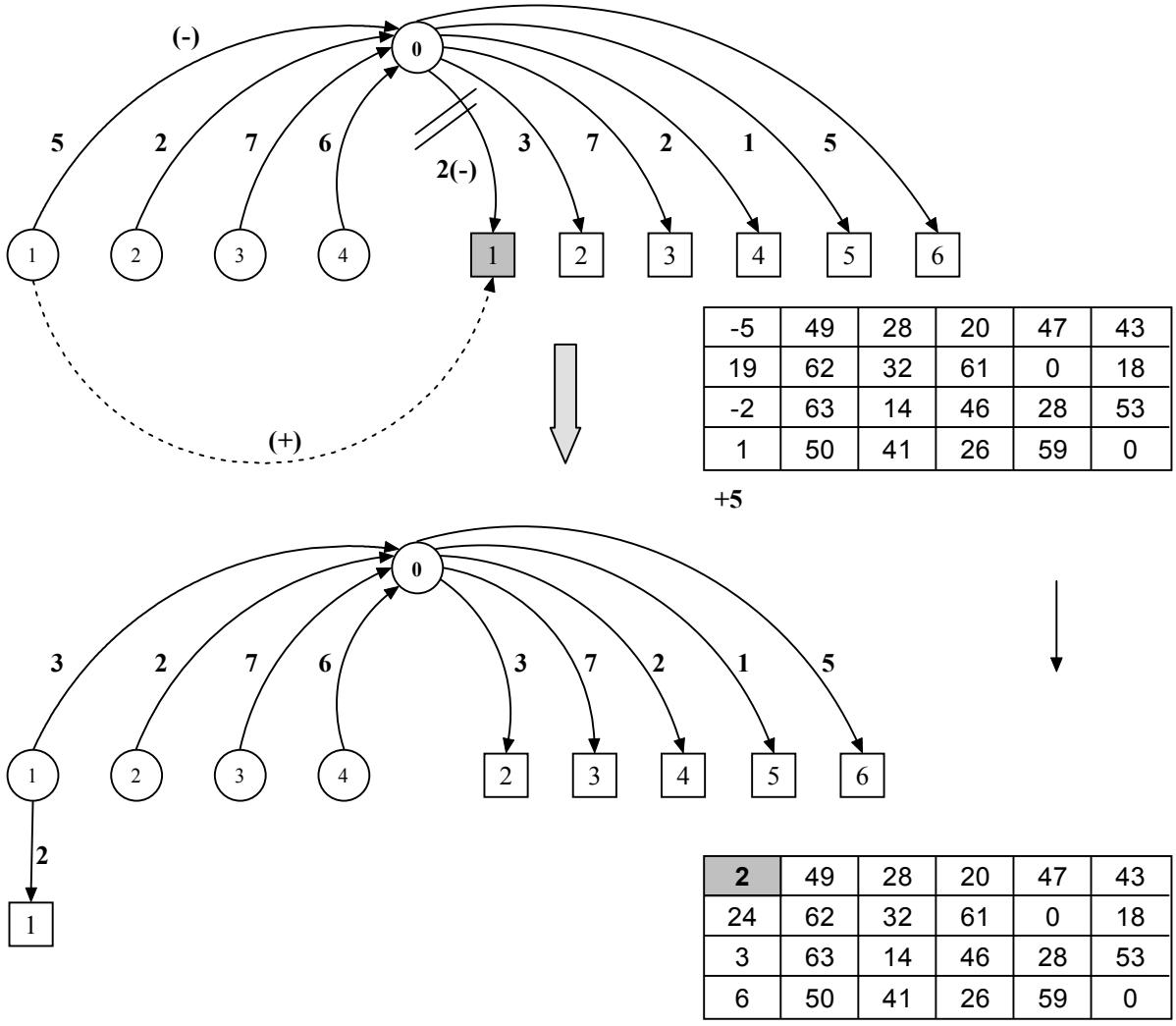


Εικόνα 5.3 – Το αρχικό δέντρου απλού ξεκινήματος

Στη συνέχεια θα παρουσιάσουμε τις επαναλήψεις του αλγορίθμου.

Επανάληψη 1

- Είναι $F^S = \{1,2,3,4\}$, $F^D = \{1,2,3,4,5,6\}$.
- $\delta = s_{gh} = \min\{s_{ij}(T) : i \in F^S, j \in F^D\} = -5 = s_{11}$, άρα $(g,h) = (1,1)$
- $\varepsilon = x_{k\ell} = \min\{x_{ij}(T) : (i,j) \in T\} = 2 = x_{01}$, άρα $(k,\ell) = (0,1)$ [1]
- Η αναλυτική ανανέωση των μεταβλητών απόφασης και μειωμένων κοστών φαίνεται στο παρακάτω σχήμα[2] [3]:



Εικόνα 5.4 – Η πρώτη επανάληψη του αλγορίθμου

Επανάληψη 2

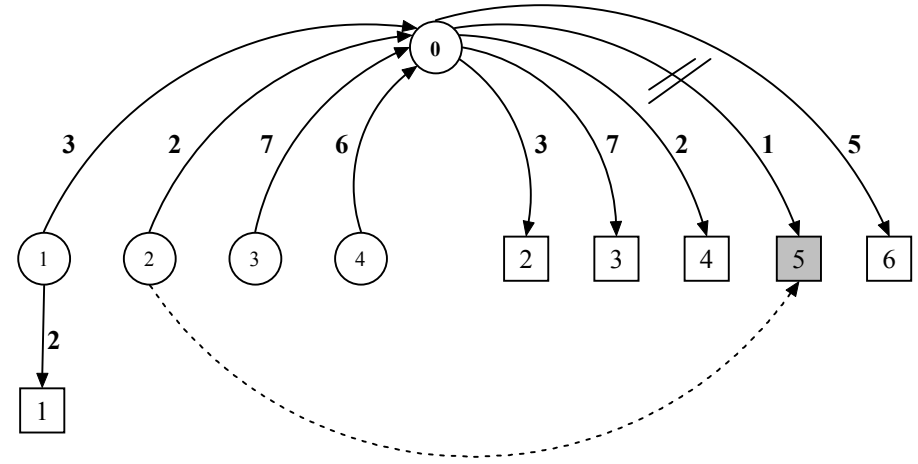
- Είναι $F^S = \{1,2,3,4\}$, $F^D = \{2,3,4,5,6\}$.

[1] Το σύνολο C^- αποτελείται από τα τεχνητά τόξα (1,0) και (0,1) και είναι x_{10}=5, x_{01}=2 , συνεπώς αφού $x_{01}<x_{10}$, το εξερχόμενο τόξο θα είναι το (0,1).

[2] Παρατηρείστε ότι στο αρχικό δάσος του προβλήματος, τα μοναδικά βασικά τόξα είναι τα τεχνητά τόξα του προβλήματος. Σταδιακά, τα τόξα αυτά θα αρχίσουν να αποκόβονται έτσι ώστε στο βέλτιστο δέντρο να μην υπάρχουν(ή αν υπάρχουν θα έχουν μηδενισμένη την αντίστοιχη μεταβλητή απόφασης).

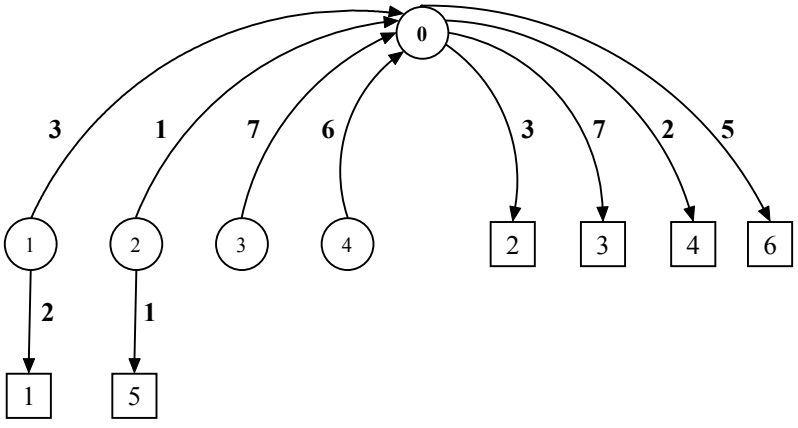
[3] Το δέντρο T^* αποτελείται σε αυτήν την επανάληψη μόνο από έναν κόμβο, τον κόμβο – στήλη 1. Έτσι, για την ανανέωση του πίνακα των μειωμένων κοστών της επανάληψης, προσθέτουμε +5 στην πρώτη στήλη του πίνακα, πράγμα που φαίνεται στην εικόνα 5.4.

- $\delta = s_{gh} = \min\left\{s_{ij}(T) : i \in F^S, j \in F^D\right\} = 0 = s_{25}$, άρα $(g,h) = (2,5)$.
- $\varepsilon = x_{k\ell} = \min\left\{x_{ij}(T) : (i,j) \in T\right\} = 1 = x_{05}$, άρα $(k,\ell) = (0,5)$.
- Η αναλυτική ανανέωση των μεταβλητών απόφασης και μειωμένων κοστών φαίνεται στο παρακάτω σχήμα:



-0

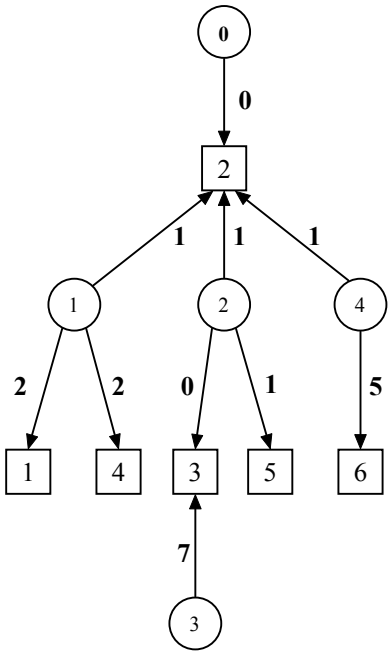
2	49	28	20	47	43
24	62	32	61	0	18
3	63	14	46	28	53
6	50	41	26	59	0



2	49	28	20	47	43
24	62	32	61	**1**	18
3	63	14	46	28	53
6	50	41	26	59	0

Εικόνα 5.5 – Η δεύτερη επανάληψη του αλγορίθμου

Δε θα παρουσιάσουμε όλες τις επαναλήψεις του αλγορίθμου, θεωρώντας ότι ο αναγνώστης έχει πλέον εξοικειωθεί με τις επαναλήψεις αλγορίθμων τέτοιου είδους. Ο αναγνώστης μπορεί ωστόσο να επιβεβαιώσει ότι στη δέκατη επανάληψη, ο αλγόριθμος παράγει το βέλτιστο δέντρο, το οποίο φαίνεται στο 5.6:



2	**1**	9	**2**	60	44
11	**1**	**0**	28	**1**	1
8	19	**7**	31	46	59
5	**1**	21	5	71	**5**

Εικόνα 5.6 – Το βέλτιστο δέντρο του προβλήματος και ο αντίστοιχος πίνακας

5.1.4 Προγραμματισμός του Αλγορίθμου

Η υλοποίηση του αλγορίθμου στο Matlab φαίνεται παρακάτω:

Αλγόριθμος Τ4

Ο παρακάτω αλγόριθμος λύνει το πρόβλημα Μεταφοράς , χρησιμοποιώντας το αλγόριθμο Παπαρρίζου με ξεκίνημα το δάσος απλού ξεκινήματος

Είσοδος: *Ο πίνακας κόστους* **C(mxn)**.

Έξοδος : *Η λύση του προβλήματος Μεταφοράς* **X(mxn)**.

```
1.  [X,S,xv]=SimpleStart(A,B,C,m,n);
2.  p=NodeFather(X,m,n);
3.  d=NodeDepth(p,m,n);
4.  while count<m+n
5.    [Fs,Fd]=SupplyDemandSets(X,m,n,p);
6.    count=0;
7.    for q=1:m+n
8.      if Fs(q)==0
9.        count=count+1;
10.     end
11. end
12.   if count<m+n
13.     t=NodeDirection(Fs,Fd,m,n,p);
14.     xv=Vectorx(B,p,X,m,n,xv,A);
15.     [min,g,h,f,Circle]=EnteringArc(S,Fs,Fd,m,n,p);
16.     [e,k,l,Cplus,Cminus]=LeavingArc(g,h,p,m,n,d,t,xv);
17.     [e1,e2,f1,z,Tcut]=Steam(p,g,h,k,l,d,m,n);
18.     [S,X]=UpdateVariables(min,e,g,h,k,l,Tcut,Cminus,Cplus,m,n,t,p,S,X);
19.     d=UpdateNodeDepth(p,m,n,Tcut,z,d,e1,e2,f1);
20.     p=UpdateNodeFather(p,z,e2,f1);
21.   end
22. end
```

Αλγόριθμος 5.1.1 – Ο ψευδοκώδικας του αλγορίθμου

Στον παραπάνω ψευδοκώδικα φαίνεται καθαρά πως γίνεται ο έλεγχος τερματισμού του αλγορίθμου. Κάθε φορά που ανανεώνονται τα σύνολα F^S και F^D (γραμμή 5), ελέγχεται αν $F^S = \varnothing$ και αναλόγως συνεχίζεται ή όχι η εκτέλεση του αλγορίθμου. Σχεδόν όλες οι συναρτήσεις που χρησιμοποιεί ο αλγόριθμος έχουν αναλυθεί διεξοδικά σε προηγούμενα κεφάλαια. Σε αυτόν τον αλγόριθμο έχουν εφαρμοστεί με τις απαραίτητες τροποποιήσεις. Θα παρουσιάσουμε μόνο τη συνάρτηση κατασκευής του δάσους απλού ξεκινήματος.

- **[X, S, xv] = SimpleStart(A, B, C, m, n)**

Αλγόριθμος SimpleStart
Ο παρακάτω αλγόριθμος υπολογίζει τις μεταβλητές που σχετίζονται με το αρχικό δάσος του απλού ξεκινήματος.
Είσοδος: *Το διάνυσμα προσφοράς* **A(m)**, *το διάνυσμα ζήτησης* **B(n)** , **C(mxn)**, **m**, **n**
Έξοδος : **X(mxn)**, **S(mxn)**, **xv(m+n)** .

```
1.  function [X,S,xv]=SimpleStart(A,B,C,m,n)
2.  for i=1:m
3.    xv(i)=A(i);
4.  end
5.  for i=1:n
6.    xv(i+m)=B(i);
7.  end
8.  for i=1:m
9.    U(i)=0;
10. end
11. for i=1:n
12.   V(i)=0;
13. end
14. for i=1:m
15.   for j=1:n
16.     X(i,j)=inf;
17.     S(i,j)=C(i,j)-U(i)-V(j);
18.   end
19. end
```

Αλγόριθμος 5.1.2 – Η κατασκευή του δάσους απλού ξεκινήματος.

Λόγω χρονικών περιορισμών, δε θα αναφερθούμε στη μαθηματική αιτιολόγηση της ορθότητας του αλγορίθμου καθώς και σε αποδείξεις των ορίων και πολυπλοκότητας. Παρόλο αυτά, η μαθηματική θεμελίωση του αλγορίθμου είναι αρκετά ενδιαφέρουσα και ο αναγνώστης μπορεί αν ανατρέξει στις εργασίες **[P1]**,**[P2]**,**[P3]**.

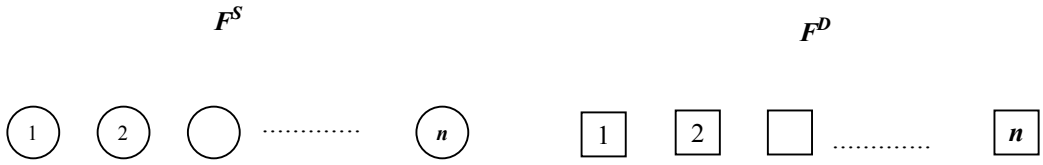
5.2 ΕΦΑΡΜΟΓΗ ΣΤΟ ΠΡΟΒΛΗΜΑ ΑΝΤΙΣΤΟΙΧΗΣΗΣ

Ο τελευταίος αλγόριθμος που θα παρουσιασθεί είναι ο αλγόριθμος Παπαρρίζου με ξεκίνημα το δάσος απλού ξεκινήματος για το πρόβλημα Αντιστοίχησης. Ο αλγόριθμος αυτός αποτελεί εξειδίκευση του αντίστοιχου αλγορίθμου για το πρόβλημα Μεταφοράς. Η εξειδίκευση αυτή πραγματοποιείται κυρίως στην επιλογή του εξερχόμενου τόξου, όπου δε χρειάζεται κάθε φορά να υπολογίζεται ο κύκλος C που σχηματίζει το εισερχόμενο τόξο (g,h) , αλλά ελέγχεται κάθε φορά ο βαθμός του κόμβου h για να καθοριστεί το εξερχόμενο τόξο (k,ℓ). Θα ξεκινήσουμε με την περιγραφή του δάσους απλού ξεκινήματος για το πρόβλημα Αντιστοίχησης.

5.2.1 Περιγραφή του Δάσους Απλού Ξεκινήματος για το πρόβλημα Αντιστοίχησης

Το δάσος απλού ξεκινήματος για το πρόβλημα Αντιστοίχησης κατασκευάζεται με τον ίδιο τρόπο που κατασκευάζεται και στο πρόβλημα Μεταφοράς. Η μόνη διαφορά είναι ότι για ένα πρόβλημα Αντιστοίχησης μεγέθους n , το δάσος του απλού ξεκινήματος θα αποτελείται

από n κόμβους γραμμές που θα ανήκουν στο σύνολο F^S και από n κόμβους στήλες που θα ανήκουν στο σύνολο F^D .(σχήμα 5.7)



Εικόνα 5.7 – Το γενικό δάσος απλού ξεκινήματος για το πρόβλημα Αντιστοίχησης

Αρχικά, οι δυϊκές μεταβλητές όλων των κόμβων του δάσους τίθενται ίσες με το μηδέν. Έτσι θα είναι

$$u_i(F) = 0,\ \forall i \in F^S$$

και

$$v_j(F) = 0,\ \forall j \in F^D$$

Συνεπώς, τα μειωμένα κόστη των τόξων (i, j) θα είναι $s_{ij}(F) = c_{ij} - u_i(F) - v_j(F) = c_{ij}$

Παράδειγμα 5.3

Δίνεται ο πίνακας κόστους

$$C = \begin{bmatrix} 3 & -3 & 19 & 10 & -27 \\ 0 & -1 & 42 & 46 & 77 \\ 70 & 32 & 43 & 71 & 90 \\ 34 & 1 & 0 & 0 & 8 \\ 1 & 7 & 34 & 65 & -7 \end{bmatrix}$$

για ένα 5×5 ***πρόβλημα Αντιστοίχησης. Να προσδιορισθεί το δάσος απλού ξεκινήματος και να υπολογισθούν τα στοιχεία,*** $u_i(F),\ i = 1..n$ **,** $v_j(F),\ j = 1..n$ ***και*** $s_{ij}(F),\ (i, j) \notin F$.

Λύση

Αρχικά θέτουμε $u_i(F) = 0,\ i = 1,...,5$ και $v_j(F) = 0,\ j = 1,...,5$. Συνεπώς, ο αρχικός πίνακας των μειωμένων κοστών θα είναι $S = C = \begin{bmatrix} 3 & -3 & 19 & 10 & -27 \\ 0 & -1 & 42 & 46 & 77 \\ 70 & 32 & 43 & 71 & 90 \\ 34 & 1 & 0 & 0 & 8 \\ 1 & 7 & 34 & 65 & -7 \end{bmatrix}$. Το αντίστοιχο δάσος απλού ξεκινήματος φαίνεται στο παρακάτω σχήμα:

Εικόνα 5.8 – Το δάσος απλού ξεκινήματος του παραδείγματος

5.2.2 Περιγραφή του Αλγορίθμου

Όπως και στο πρόβλημα Μεταφοράς, έτσι και στο πρόβλημα Αντιστοίχησης ο αλγόριθμος δουλεύει με δέντρα. Έτσι μετατρέπουμε το δάσος της εικόνας 5.7 σε δέντρο προσθέτοντας $2n$ τεχνητά(*artificial*) τόξα και έναν ουδέτερο(*neutral*) κόμβο 0. Ο κόμβος 0 είναι πλέον η ρίζα του δέντρου. Το δέντρο T που προκύπτει συνδέεται με τα σύνολα F^S και F^D με την παρακάτω σχέση:

$$T = F \cup \{(r,0),\ \forall T_r \in F^S\} \cup \{(0,w),\ \forall T_w \in F^D\}$$

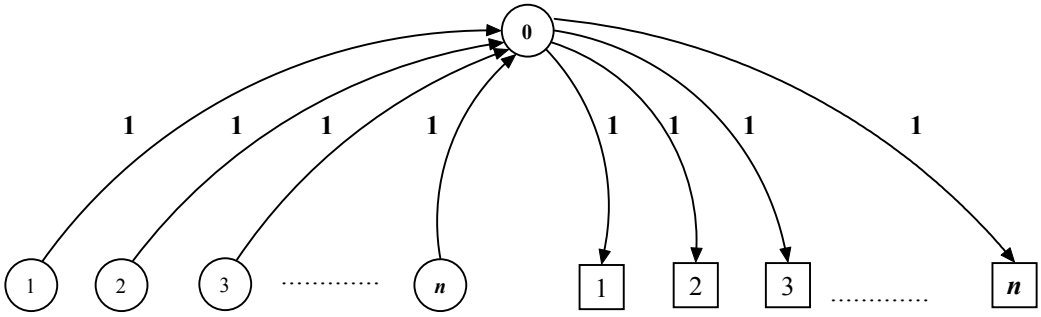
όπου

$$F^S = \{T_i : i \in I\}$$

και

$$F^D = \{T_j : j \in J\}$$

όπου I, J είναι τα σύνολα των γραμμών και στηλών του προβλήματος. Οι μεταβλητές απόφασης των τεχνητών τόξων αρχικά τίθενται ίσες με τη μονάδα. Θα δούμε ότι κατά τη διάρκεια της εκτέλεσης του αλγορίθμου οι μεταβλητές απόφασης των τεχνητών τόξων που περιέχουν τον κόμβο j-είτε αυτός είναι κόμβος του F^S είτε του F^D- έχουν πάντα θετικές τιμές και ίσες με $|d_j - 2|$, όπου d_j ο βαθμός του κόμβου j. Το δέντρο T που προκύπτει θα είναι της μορφής:



Εικόνα 5.9 – Το γενικό δέντρο που προκύπτει από το δάσος απλού ξεκινήματος

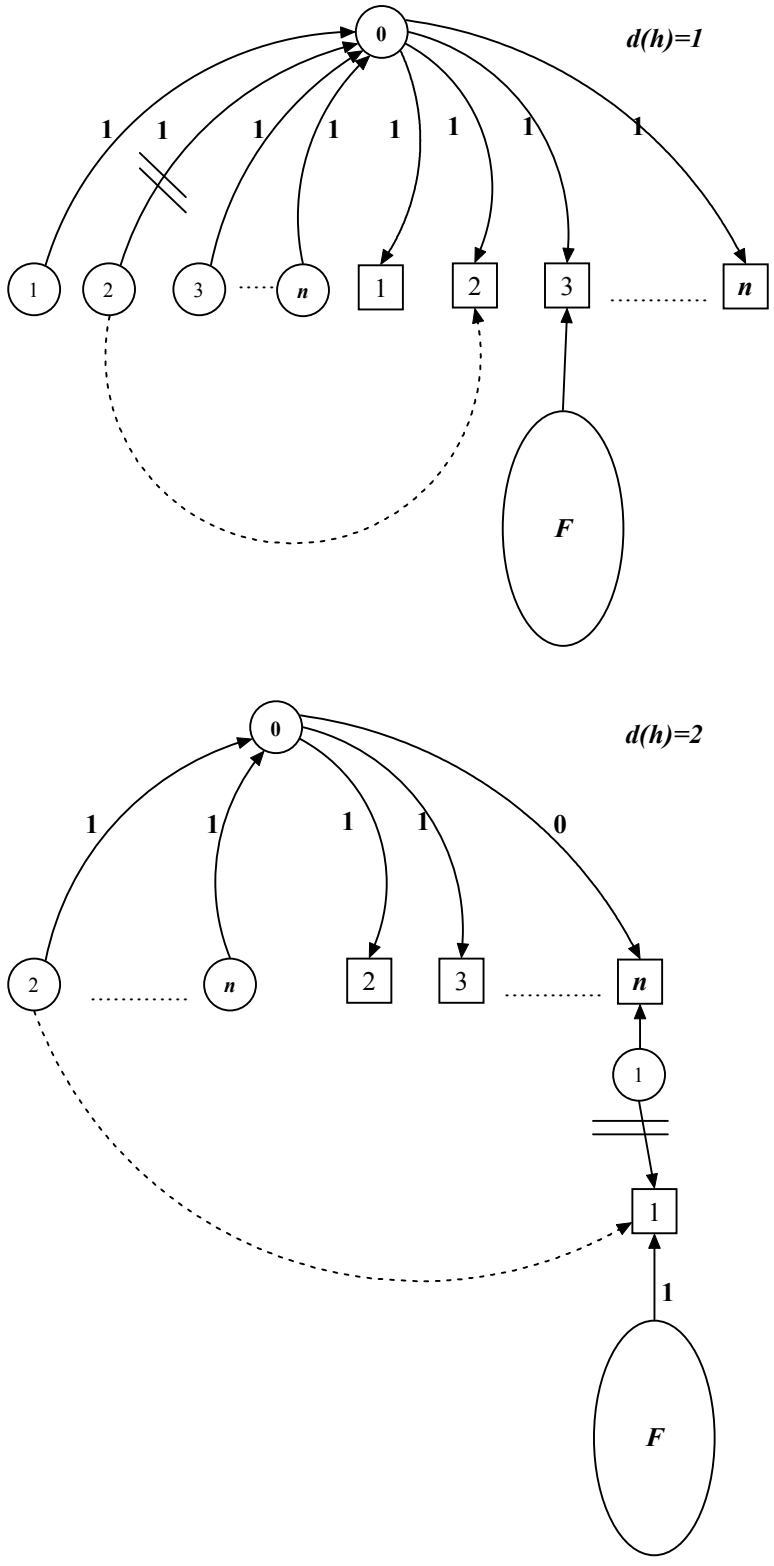
Ο αλγόριθμος επιλέγει το εισερχόμενο τόξο (g,h) με βάση τη σχέση:

$$\delta = s_{gh} = \min\{s_{ij}(T) : i \in F^S, j \in F^D\}$$

Το εξερχόμενο τόξο (k,ℓ) επιλέγεται με βάση το βαθμό του κόμβου στήλη h. Διακρίνουμε λοιπόν τις εξής περιπτώσεις. Αν $d(h) = 2$, τότε εξερχόμενο είναι το μοναδικό προς τα κάτω τόξο (i,h) του δέντρου T. Αν τώρα $d(h) = 1$, όπου f είναι ο κόμβος του συνόλου F^S που ανήκει στον κύκλο C και είναι προσαρτημένος στη ρίζα του δέντρου T, τότε εξερχόμενο θα είναι το τόξο $(f,0)$.

Παρατηρείστε ότι κατά την επιλογή του εξερχόμενου τόξου, έχουμε να εξετάσουμε δύο περιπτώσεις μόνον. Ο βαθμός του κόμβου f στην περίπτωση που $d(h) = 1$ δε μας ενδιαφέρει, πράγμα που δε συνέβαινε στους αλγορίθμους που εξετάστηκαν στα κεφάλαια 3

και 4, αφού έπρεπε να εξετάσουμε ξεχωριστά δύο υποπεριπτώσεις. Στο παρακάτω σχήμα φαίνονται καθαρά οι δύο περιπτώσεις επιλογής του εξερχόμενου τόξου (Παρατηρείστε ότι δεν είναι οι ίδιες περιπτώσεις με τον αλγόριθμο που παρουσιάσθηκε στο Κεφάλαιο 4).



Εικόνα 5.10 – Οι δύο περιπτώσεις επιλογής του εξερχόμενου τόξου

Τα μειωμένα κόστη ανανεώνονται με βάση το αποκομμένο δέντρο $T*$ με τον τρόπο που είδαμε και στους άλλους αλγορίθμους(ανανεώνονται τα μειωμένα κόστη των τόξων (i, j)

τέτοια ώστε $i \in T^* \vee j \in T^*$). Στη συνέχεια ακολουθεί η βηματική περιγραφή του αλγορίθμου.

Ο αλγόριθμος τερματίζει και παράγει το βέλτιστο δέντρο όταν $F^S = \varnothing$.

5.2.3 Βηματική Περιγραφή του Αλγορίθμου

Στη συνέχεια θα παρουσιάσουμε τον αλγόριθμο σε μορφή βημάτων και βασισμένοι στα βήματα του αλγορίθμου θα λύσουμε ένα πρόβλημα.

ΒΗΜΑ 0:*(Καθορισμός αρχικών μεταβλητών)*
Υπολόγισε το δάσος απλού ξεκινήματος με τη μέθοδο που περιγράφηκε. Πρόσθεσε έναν ουδέτερο κόμβο 0 και τεχνητά τόξα της μορφής $(i,0)$ αν $i \in I$ και $(0,j)$ αν $j \in J$. Έτσι κατασκευάζεται το αρχικό εφικτό δέντρο T του προβλήματος.

ΒΗΜΑ 1:*(Έλεγχος βελτιστότητας)*

ΒΗΜΑ 2:*(Επιλογή εισερχομένου τόξου)*
Υπολόγισε το εισερχόμενο τόξο (g,h) ελέγχοντας τα μειωμένα κόστη των μη βασικών τόξων με βάση τον τύπο $s \quad \min\left\{s_{ij}(T) : i \in F^S, j \in F^D\right\}$.

ΒΗΜΑ 3:*(Επιλογή εξερχόμενου τόξου)*

είναι ο κόμβος του συνόλου ^{S} που ανήκει στον κύκλο C και είναι προσαρτημένος στη ρίζα του δέντρου T.

ΒΗΜΑ 4:*(Ανανέωση των μεταβλητών)*
Υπολόγισε το δέντρο $*$ που είναι το δέντρο που αποκόβεται από τη ρίζα όταν αφαιρείται

ο κόμβος $b \in T$ είναι κόμβος-γραμμή θέσε $s_{bj}(T') = s_{bj}(T) - q$, $j = 1,...,n$ αλλιώς (δηλαδή ο κόμβος $b \in T^*$ είναι κόμβος στήλη θέσε $s_{ib}(T') = s_{ib}(T) + q$, $i = 1,...,n$. Θέσε τις μεταβλητές απόφασης των τεχνητών τόξων που περιέχουν τον κόμβο j ίσες με $\left|d_j - 2\right|$.

ΒΗΜΑ 5:*(Ανανέωση του τρέχοντος δέντρου)*

Παράδειγμα 5.4
Δίνεται ο πίνακας κόστους

$$C = \begin{bmatrix} 3 & -3 & 19 & 10 & -27 \\ 0 & -1 & 42 & 46 & 77 \\ 70 & 32 & 43 & 71 & 90 \\ 34 & 1 & 0 & 0 & 8 \\ 1 & 7 & 34 & 65 & -7 \end{bmatrix}$$

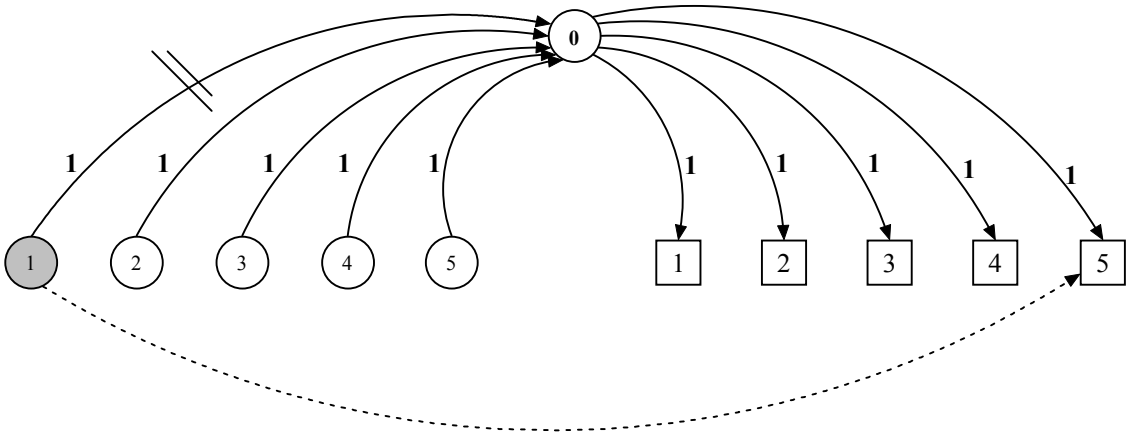
για ένα 5×5 ***πρόβλημα Αντιστοίχησης. Να λυθεί το πρόβλημα χρησιμοποιώντας τον αλγόριθμο Παπαρρίζου με απλό ξεκίνημα.***

Λύση

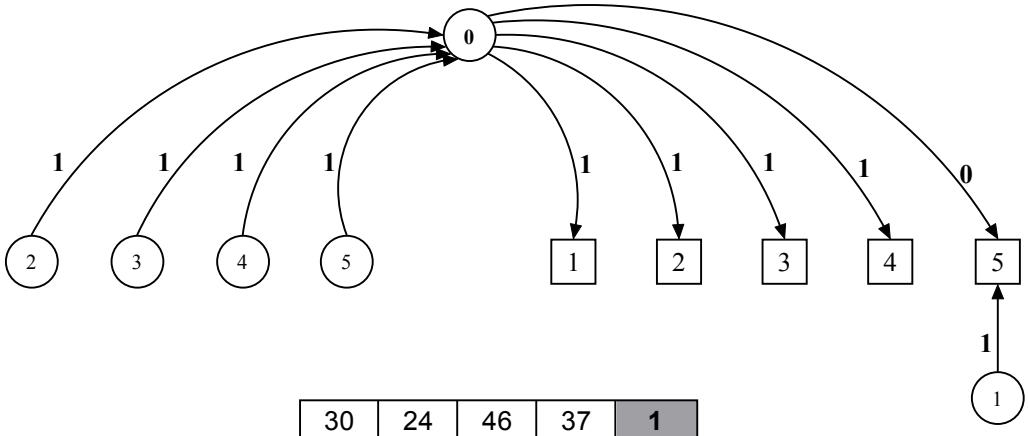
Τα δεδομένα του παραδείγματος είναι ίδια με τα δεδομένα του παραδείγματος 5.3. Συνεπώς, αφού έχουμε υπολογίσει το δάσος απλού ξεκινήματος, προχωράμε στην πρώτη επανάληψη του αλγορίθμου.

Επανάληψη 1

- Είναι $F^S=\{1,2,3,4,5\}$, $F^D=\{1,2,3,4,5\}$ άρα $\delta=\min\{s_{ij}(T): i\in F^S, j\in F^D\}=s_{51}$, συνεπώς $(g,h)=(5,1)$ και $\delta=-27$.



3	-3	19	10	-27	+27
0	-1	42	46	77	
70	32	43	71	90	
34	1	0	0	8	
1	7	34	65	-7	



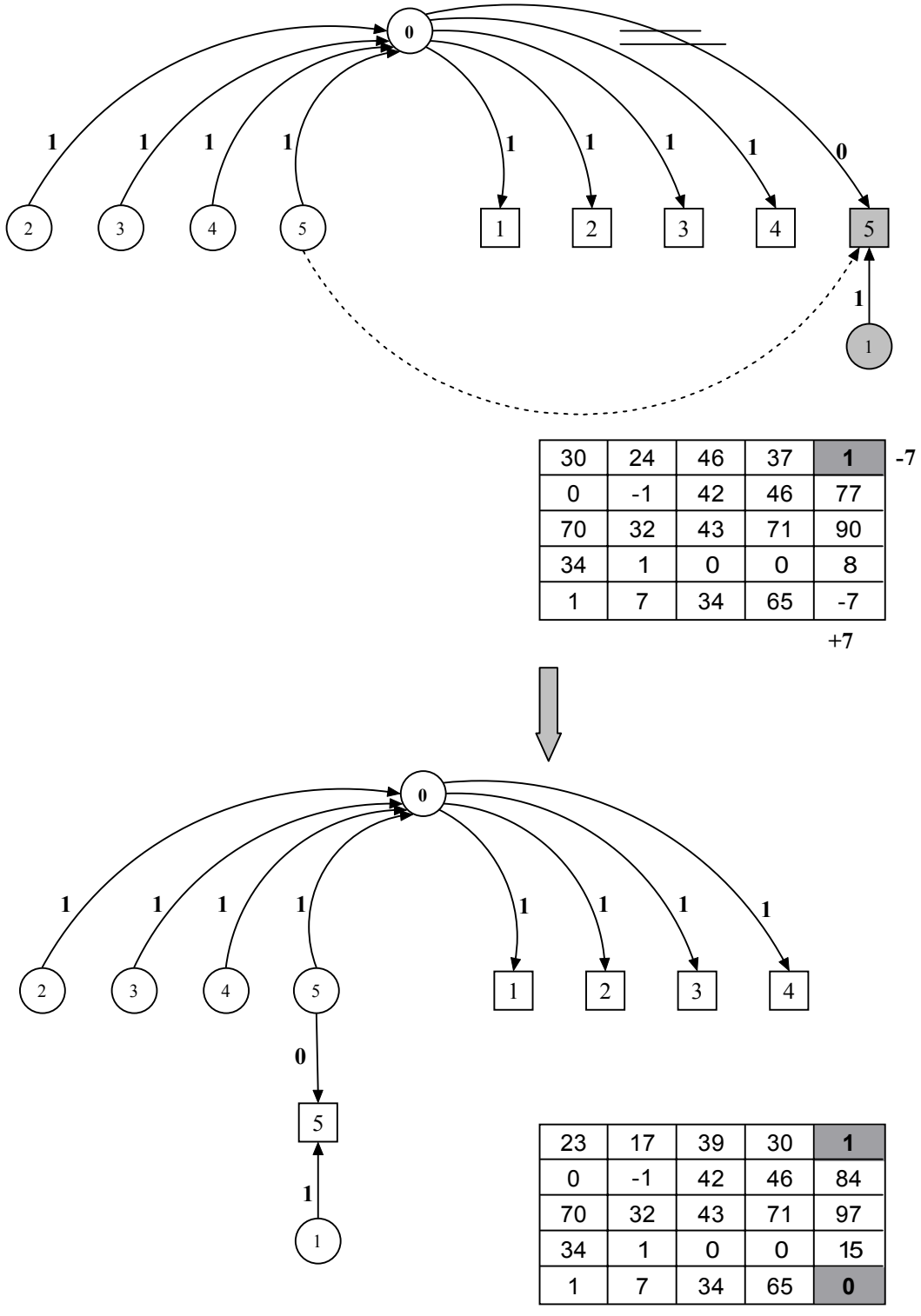
30	24	46	37	**1**
0	-1	42	46	77
70	32	43	71	90
34	1	0	0	8
1	7	34	65	-7

Εικόνα 5.11 – Η πρώτη επανάληψη του αλγορίθμου

- Είναι $d(h) = d(5) = 1$, άρα $(k, \ell) = (f, 0) = (1, 0)$.
- Η ανανέωση των δομών και μεταβλητών του αλγορίθμου φαίνεται στο σχήμα 5.11.

Επανάληψη 2

- Είναι $F^S = \{2,3,4,5\}$, $F^D = \{1,2,3,4,5\}$ άρα $\delta = \min\{s_{ij}(T) : i \in F^S, j \in F^D\} = s_{55}$, συνεπώς $(g,h) = (5,5)$ και $\delta = -7$.

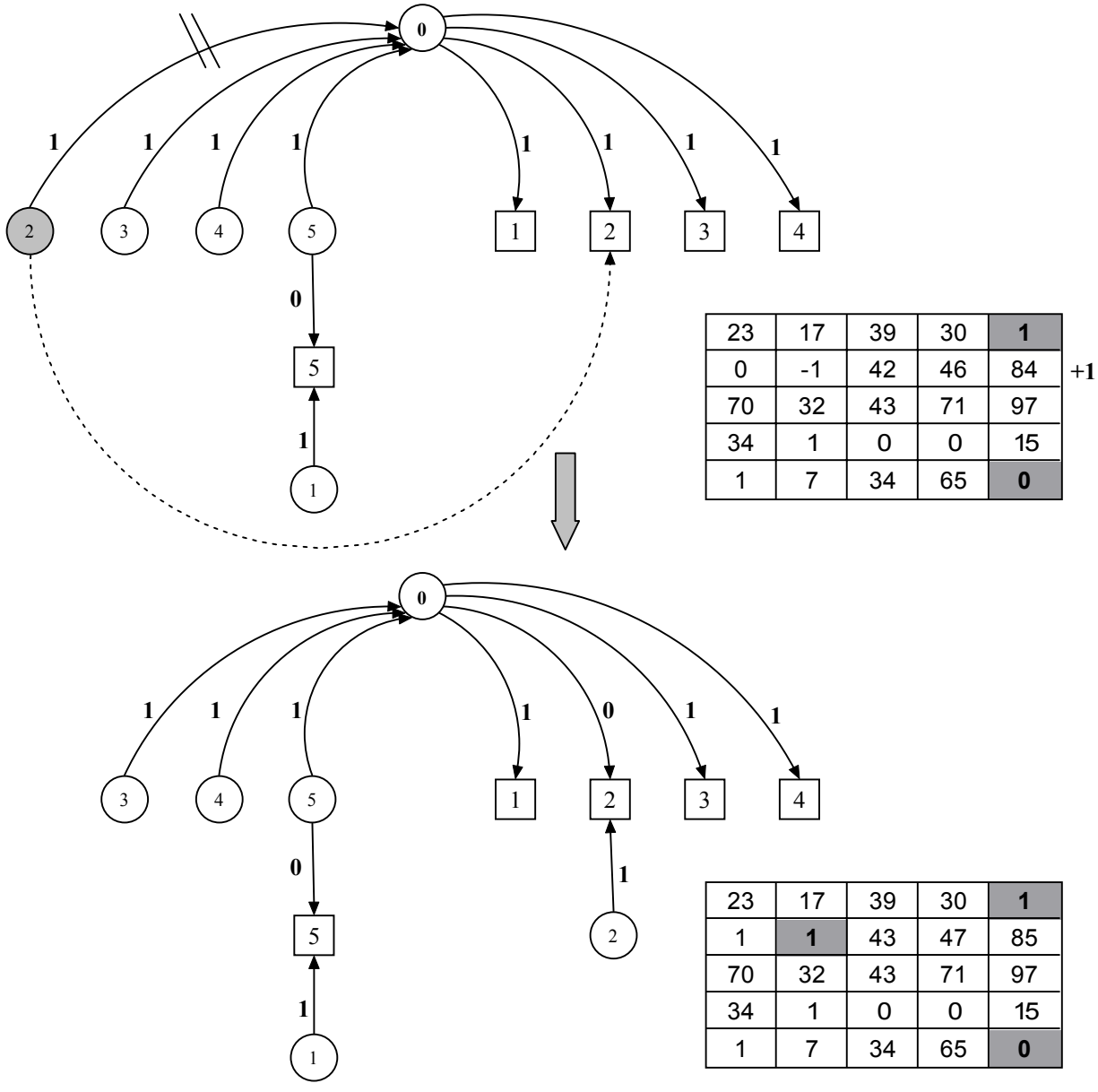


Εικόνα 5.12 – Η δεύτερη επανάληψη του αλγορίθμου

- Είναι $d(h) = d(5) = 2$, άρα $(k, \ell) = (i, h) = (0,5)$.
- Η ανανέωση των δομών και μεταβλητών του αλγορίθμου φαίνεται στο σχήμα 5.12.

Επανάληψη 3

- Είναι $F^S = \{1,2,3,4,5\}$, $F^D = \{1,2,3,4\}$, $\delta = \min\{s_{ij}(T) : i \in F^S, j \in F^D\} = s_{22} = -1$, συνεπώς $(g,h) = (2,2)$.

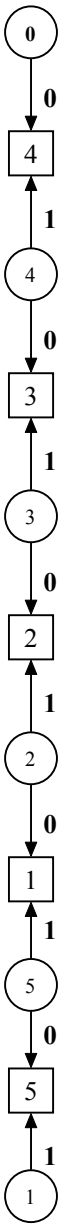


Εικόνα 5.13 – Η τρίτη επανάληψη του αλγορίθμου

- Είναι $d(h) = d(2) = 1$, άρα $(k, \ell) = (f, 0) = (2,0)$.
- Η ανανέωση των δομών και μεταβλητών του αλγορίθμου φαίνεται στο σχήμα 5.13.

Ο αναγνώστης μπορεί να επιβεβαιώσει ότι στην 9η επανάληψη του αλγορίθμου θα προκύψει κενό σύνολο F^S, πράγμα που σημαίνει ότι ο αλγόριθμος τερματίζει και παράγει το βέλτιστο δέντρο με τιμή αντικειμενικής συνάρτησης $z^* = \sum_{(i,j) \in T} c_{ij} x_{ij} = 16$.

Στο παρακάτω σχήμα βλέπουμε το βέλτιστο δέντρο του αλγορίθμου για το πρόβλημα του παραδείγματος 5.4.



22	17	28	19	**1**
0	**1**	32	36	85
37	**0**	**1**	28	65
44	12	**0**	**1**	26
1	7	23	54	**0**

Εικόνα 5.14 – Το βέλτιστο δέντρο του αλγορίθμου και ο αντίστοιχος πίνακας

5.2.5 Προγραμματισμός του Αλγορίθμου

Παρακάτω παρουσιάζουμε τον ψευδοκώδικα του αλγορίθμου σε μορφή κλήσεως συναρτήσεων. Επίσης θα παρουσιαστούν και θα αναλυθούν οι σημαντικότερες συναρτήσεις που χρησιμοποιήθηκαν και αυτές που έχουν κάποια ιδιαιτερότητα που αφορά τη φιλοσοφία του αλγορίθμου. Οι βασικές συναρτήσεις του αλγορίθμου, όπως αυτήν της ανανέωσης βάθους είναι ίδιες με αυτές που χρησιμοποιήθηκαν και στο ν αλγόριθμο Simplex.

Αλγόριθμος Α4
Ο παρακάτω αλγόριθμος λύνει το πρόβλημα Αντιστοίχησης , χρησιμοποιώντας το αλγόριθμο Παπαρρίζου με ξεκίνημα το δάσος απλού ξεκινήματος
Είσοδος: *Ο πίνακας κόστους* **C(mxn)**.
Έξοδος : *Η λύση του προβλήματος Μεταφοράς* **X(mxn)**.

```
1.  [X,S,xv]=SimpleStart(C,n);
2.  p=NodeFather(X,n);
3.  d=InitializeNodeDepth(p,n);
4.  while count<2*n
5.    [Fs,Fd]=SupplyDemandSets(X,n,p);
6.    count=0;
7.    for q=1:2*n
8.      if Fs(q)==0
9.        count=count+1;
10.     end
11.   end
12.   if count<2*n
13.     [min,g,h,f,Circle]=EnteringArc(S,Fs,Fd,n,p);
14.     xv=ArcHelp(Fs,Fd,p,n,xv);
15.     deg=degree(X,xv,n);
16.     [k,l,xv,p,upwards,Tcut]=LeavingArc(g,h,deg,f,Circle,p,n,xv);
17.     [e1,e2,f1,z]=Steam(p,g,h,k,l,d,n,Tcut);
18.     d=UpdateNodeDepth(p,n,Tcut,z,d,e1,e2,f1);
19.     p=UpdateNodeFather(p,z,e2,f1);
20.     [S,X]=UpdateVariables(g,h,k,l,d,upwards,n,p,X,S,Tcut,min,Fs);
21.   end
22. end
```

Αλγόριθμος 5.2.1 – Επίλυση προβλήματος Αντιστοίχησης με το δάσος απλού ξεκινήματος

- **[k, l, xv, p, upwards, Tcut]=LeavingArc(g, h, deg, f, Circle, p, n, xv)**

Αλγόριθμος LeavingArc
Υπολογίζει το εξερχόμενο τόξο με βάση το βαθμό του κόμβου στήλη h.
Είσοδος: g, h, deg(n), f, Circle(), p(2n), n, xv(2n)
Έξοδος : k, l, xv(2n), p(2n), upwards, Tcut()

```
1.  function [k,l,xv,p,upwards,Tcut]=LeavingArc(g,h,deg,f,Circle,p,n,xv)
2.  if deg(h+n)==2
3.    k=p(h+n);   l=h;   start=l+n;
4.    Tcut=Preorder(start,p,n);
5.    if p(h+n)==0
6.      xv(h+n)=inf;
7.    end
8.    upwards=0;
9.  end
10. if deg(h+n)==1
11.   k=f;
12.   l=0;
13.   start=f;
14.   Tcut=Preorder(start,p,n);
15.   xv(f)=inf;
16.   upwards=1;
17.   i=1;
18.   while Circle(i)~=h+n
19.     i=i+1;
20.   end
21.   if p(h+n)==0
22.     xv(h+n)=0;
23.   else
24.     j=i;
25.     while Circle(j)~=0
26.       j=j-1;
27.     end
28.     xv(Circle(j)-n)=0;
29.   end
30. end
```

Αλγόριθμος 5.2.2 – Επιλογή εξερχομένου τόξου

Παρατηρούμε στον παραπάνω κώδικα ότι γίνεται ο έλεγχος του βαθμού του κόμβου h και αναλόγως επιλέγεται το εξερχόμενο τόξο.

5.3 ΑΝΑΦΟΡΕΣ

[P1] K. Paparrizos, *"A non improving simplex algorithm for transportation problems"*, Operations Research, vol 30, pp 1-15, 1996

[P2] H. Achatz, P. Kleinschmidt and K. Paparrizos, "*A dual forest algorithm for the assignment problem,*" Dimacs, vol. 4, pp. 1-10, 1990.

[P3] K. Paparrizos, *"An Infeasible (Exterior Point) Simplex Algorithm for Assignment Problems"*, Mathematical Programming 51 (1991) 45-54 North Holland

[Cun] Cunningham, W.H. (1976),"*A Network Simplex Method"*, Math Prog 11,105-116

[CLR] Cormen , Leiserson , Rivest, *"Introduction to Algorithms"* , MIT Press, (1990)

[JB] Jensen and Barnes, (1980),*"Network Flow Programming"*, Malabar, Fla.: R.E. Krieger Pub. Co., 1980

[Pap1] K. Paparrizos, *"Linear Programming – Algorithms and Applications"*, in Greek

[Ber] Bertsekas D.**, "***Network Optimization: Continuous and Discrete Models"*, Athena Scientific, 1998

[Kn] D.E. Knuth, *"The Art of Computer Programming – Fundamental Algorithms"*, 3rd edition, Addison-Wesley, 1997

[Roc] Rockafellar R.T.,**"***Network Flows and Monotropic Optimization"*, Wiley-Interscience, 1984 (610 pp.).

[Pap2] K. Paparrizos, *"Network Programming"*, Lecture notes in Greek

[Pap3] K. Paparrizos, *"Algorithms – Analysis, Design and Complexity"*, Lecture notes in Greek

[SS] Stephanides G., Samaras N., *"Computational Methods with Matlab"*, Zygos Publications, Thessaloniki 1999, in Greek

[PK] Paparrizos K., *"Matlab 6"*, Zygos Publications, Thessaloniki 2001, in Greek

ΚΕΦΑΛΑΙΟ 6

ΥΠΟΛΟΓΙΣΤΙΚΗ ΜΕΛΕΤΗ

6.1 ΕΙΣΑΓΩΓΗ

Στο κεφάλαιο αυτό παρουσιάζεται μια αρκετά εκτενής υπολογιστική μελέτη (*computational study*) που πραγματοποιήθηκε με τους τέσσερις αλγορίθμους που αναπτύχθηκαν. Είναι ίσως το πιο σημαντικό κεφάλαιο της διπλωματικής εργασίας αφού παρουσιάζει όλα τα αποτελέσματα που προκύπτουν από την εκτέλεση των αλγορίθμων σε ένα συγκεκριμένο υπολογιστικό σύστημα. Τα προβλήματα που χρησιμοποιήθηκαν είναι διαφόρων διαστάσεων και κλάσεων[1], οι οποίες θα αναλυθούν παρακάτω.

Υπολογιστικές μελέτες χρησιμοποιούνται συχνά για την σύγκριση αλγορίθμων και την εξαγωγή συμπερασμάτων. Μια υπολογιστική μελέτη μπορεί να ορισθεί ως η σύγκριση δύο ή περισσοτέρων αλγορίθμων σε κοινά προβλήματα Οι σημαντικότερες έννοιες που μας ενδιαφέρουν σε μια υπολογιστική μελέτη είναι ο αριθμός των επαναλήψεων του αλγορίθμου (*number of iterations*) και ο συνολικός χρόνος της κεντρικής μονάδας επεξεργασίας (*CPU time*) **[Sam]**. Οι δύο αυτές ποσότητες δεν είναι αναγκαστικά ανάλογες. Υπάρχει περίπτωση κατά τη σύγκριση δύο αλγορίθμων *a, b,* ενώ ο *a* να κάνει τις μισές επαναλήψεις από τον *b*, να απαιτεί περισσότερο χρόνο σε CPU από τον *b*. Αυτό συμβαίνει επειδή ο *a* θα εκτελεί πολύ πιο χρονοβόρες επαναλήψεις από τον *b*.

Στα επόμενα κεφάλαια της εργασίας, θα αναφερόμαστε στον καθένα αλγόριθμο ξεχωριστά με τα ακρωνύμια που φαίνονται στον παρακάτω πίνακα.

Πίνακας 6.1 – Συντομογραφίες Ονομάτων Αλγορίθμων

Ακρωνύ-μιο	**Αλγόριθμος**
T1	Πρωτεύων Αλγόριθμος Simplex για το πρόβλημα Μεταφοράς
A1	Πρωτεύων Αλγόριθμος Simplex για το πρόβλημα Αντιστοίχησης
T2	Αλγόριθμος Παπαρρίζου με ξεκίνημα το δέντρο Balinski για το πρόβλημα Μεταφοράς
A2	Αλγόριθμος Παπαρρίζου με ξεκίνημα το δέντρο Balinski για το πρόβλημα Αντιστοίχησης
T3	Αλγόριθμος Παπαρρίζου με ξεκίνημα το δάσος AKP για το πρόβλημα Μεταφοράς
A3	Αλγόριθμος Παπαρρίζου με ξεκίνημα το δάσος AKP για το πρόβλημα Αντιστοίχησης
T4	Αλγόριθμος Παπαρρίζου με απλό ξεκίνημα για το πρόβλημα Μεταφοράς
A4	Αλγόριθμος Παπαρρίζου με απλό ξεκίνημα για το πρόβλημα Αντιστοίχησης

6.2 ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΗ ΕΜΠΕΙΡΙΑ

Και οι τέσσερις αλγόριθμοι που παρουσιάσθηκαν προγραμματίστηκαν σε Matlab 6. Ο προγραμματισμός ακολουθεί κατά αυστηρό τρόπο την περιγραφή των αλγορίθμων. Οι ολοκληρωμένοι κώδικες είναι διαθέσιμοι σε ξεχωριστό ένθετο. Οι αλγόριθμοι προγραμματίστηκαν έτσι ώστε να λύνουν μόνο αραιά προβλήματα, δηλαδή προβλήματα που περιγράφονται από διμερή γραφήματα $G(S,D,E)$ όπου επιτρέπονται όλα τα δυνατά τόξα (i,j) ώστε $i \in S \wedge j \in D$.[2] Ο αποτελεσματικός προγραμματισμός των αλγορίθμων για αραιά προβλήματα απαιτεί τη χρησιμοποίηση άλλων δομών δεδομένων (π.χ σωροί Fibonacci **[P1]**) και πρόκειται να πραγματοποιηθεί σε μελλοντικό στάδιο.

Εδώ πρέπει να αναφερθεί ότι ο προγραμματισμός έγινε με τον ίδιο ακριβώς τρόπο και για τους τέσσερις αλγορίθμους. Αυτό σημαίνει ότι οι συναρτήσεις που χρησιμοποιήθηκαν

[1]Με τον όρο «κλάση» ενός προβλήματος εννοούμε ορισμένες μαθηματικές ιδιότητες που χαρακτηρίζουν τη μήτρα δεδομένων, π.χ. ομοιόμορφη κατανομή των στοιχείων , οι οποίες συνήθως επηρεάζουν την απόδοση του αλγορίθμου.

[2]Στην περίπτωση ενός πυκνού διμερούς δικτύου $G(S,D,E)$ θα ισχύει $card(E) = card(S)card(D)$, διαφορετικά, το δίκτυο είναι αραιό με πυκνότητα $d = \frac{card(E)}{card(S)card(D)} \in (0,1)$.

γράφτηκαν με βάση τις ίδιες προγραμματιστικές τεχνικές για όλους τους αλγορίθμους. Απλώς έγινε τροποποίηση αναλόγως με τα ιδιαίτερα χαρακτηριστικά των αλγορίθμων. Αυτό είναι απαραίτητο για να επιτευχθεί μια δίκαια υπολογιστική μελέτη**[PPS]**. Οι κλάσεις των δεδομένων που χρησιμοποιήθηκαν θα περιγραφούν παρακάτω.

Η ανάπτυξη του λογισμικού πέρασε από δύο φάσεις. Στην πρώτη φάση, ο προγραμματισμός έγινε με βάση την ορθότητα των αποτελεσμάτων. Δε λήφθηκαν υπ' όψιν τεχνικές και μέθοδοι για την υπολογιστική αποτελεσματικότητα των αλγορίθμων. Μια προκαταρτική υπολογιστική μελέτη έφερε αποτελέσματα που δεν συμβάδιζαν με το θεωρητικό υπόβαθρο των αλγορίθμων. Συνεπώς, έγινε προσπάθεια να χρησιμοποιηθούν προχωρημένες προγραμματιστικές τεχνικές για την αποδοτικότερη ανανέωση των μεταβλητών του αλγορίθμου. Αυτή η προσπάθεια απέδωσε καρπούς και παρήγαγε πολύ καλά αποτελέσματα τα οποία θα παρουσιασθούν διεξοδικά.

Στην πρώτη φάση ανάπτυξης του λογισμικού , η συνάρτηση Preorder(), που χρησιμοποιείται κατά κόρον από όλους τους αλγορίθμους είχε προγραμματιστεί μη αναδρομικά. Χρησιμοποιώντας ένα ειδικό εργαλείο του Matlab, τον Matlab Profiler, καταλήξαμε στο συμπέρασμα ότι περίπου 50% του χρόνου εκτέλεσης του αλγορίθμου καταναλωνόταν από την μη αναδρομική συνάρτηση Preorder(). Αυτό είχε ως αποτέλεσμα τον αναδρομικό προγραμματισμό της συνάρτησης. Ο αλγόριθμος της αναδρομικής συνάρτησης `preorder(p()`[3]`,root)` είναι ο παρακάτω:

Αλγόριθμος **`preorder(p(),root)`**

Βήμα 1
Επίσκεψη του κόμβου **`root`**. Θέσε **`y = y U root`**
Βήμα 2
Για κάθε υπόδεντρο ρίζας **`t`** τέτοιο ώστε **`p(t)==root`** , κάλεσε την **`preorder(p(),t)`**

Το διάνυσμα της διάσχισης που προκύπτει θα είναι το διάνυσμα **`y`**.

Για να κατανοηθεί η λειτουργία της αναδρομικής συνάρτησης θα παρουσιάσουμε ένα παράδειγμα. Έστω λοιπόν ότι λύνουμε ένα πρόβλημα αντιστοίχησης μεγέθους 7 με τον αλγόριθμο Α3. Ας θεωρήσουμε ότι η τρέχουσα κατάσταση του δέντρου του αλγορίθμου περιγράφεται από το παρακάτω διάνυσμα πατέρα – κόμβου:

$$p = \begin{bmatrix} 13 & 10 & 11 & 9 & 10 & 8 & 12 & 0 & 7 & 0 & 5 & 3 & 2 & 0 \end{bmatrix}$$

Έστω αποκόβεται το τόξο $(k,\ell) = (3,0)$. Χρησιμοποιώντας το διάνυσμα p, μπορούμε να ζωγραφίσουμε το δέντρο και να διαπιστώσουμε ότι το αποκομμένο δέντρο $T*$ έχει ρίζα τον κόμβο – στήλη 3 (κόμβος 3+7=10). Ο αναδρομικός υπολογισμός του διανύσματος διάσχισης y φαίνεται παρακάτω.

- Θέτουμε $y(1) = 10$.
- Τα παιδιά του κόμβου 10 είναι το σύνολο $C = \{2,5\}$.
- Θέτουμε $y(2) = 2$. Τα παιδιά του κόμβου 2 είναι το σύνολο $C = \{13\}$.
- Θέτουμε $y(3) = 13$. Τα παιδιά του κόμβου 13 είναι το σύνολο $C = \{1\}$.
- Θέτουμε $y(4) = 1$. Ο κόμβος 1 δεν έχει παιδιά. Όλοι οι κόμβοι που ανήκουν στο υπόδεντρο που είναι ριζωμένο στον κόμβο 2 έχουν επισκεφτεί. Συνεπώς, ελέγχουμε το άλλο κλαδί του δέντρου.

[3] Πρόκειται για το διάνυσμα πατέρα – κόμβου *p*

- Θέτουμε $y(5)=5$ και ακολουθούμε την ίδια αναδρομική διαδικασία για το υπόδεντρο που είναι ριζωμένο στον κόμβο 5.

Μετά τον τερματισμό της αναδρομικής διαδικασίας (έλεγχος όλων των κλαδιών της αρχικής ρίζας) το διάνυσμα διάσχισης που παίρνουμε είναι το

$$y=\begin{bmatrix}10 & 2 & 13 & 1 & 5 & 11 & 3 & 12 & 7 & 9 & 4\end{bmatrix}$$

Μεγάλη υπολογιστική βελτίωση προήλθε επίσης και από την εφαρμογή της διαδικασίας ανανέωσης των βαθών των κόμβων όπως αυτήν περιγράφηκε στο κεφάλαιο 2.

6.3 ΟΙ ΚΛΑΣΕΙΣ ΤΩΝ ΔΕΔΟΜΕΝΩΝ

Στη συνέχεια θα περιγράψουμε τις κλάσεις των δεδομένων που χρησιμοποιήσαμε για τη διεξαγωγή των υπολογιστικών μελετών.

6.3.1 Οι κλάσεις στο πρόβλημα Μεταφοράς

Οι $m\times n$ πίνακες δεδομένων c_{ij} καθώς και τα διανύσματα προσφοράς και ζήτησης που χρησιμοποιήσαμε για τη διεξαγωγή της υπολογιστικής μελέτης ανήκουν στις παρακάτω κλάσεις:

Class 1
Ασσυμετρικοί πίνακες πραγματικών αριθμών ομοιόμορφα κατανεμημένων στο διάστημα $[1,100)$. Δηλαδή $c_{ij}\sim U(1,100),\ i=1,...,m,\ j=1,...,n$.

Class 2
Ασσυμετρικοί πίνακες πραγματικών αριθμών ομοιόμορφα κατανεμημένων στο διάστημα $[1,1000000)$. Δηλαδή $c_{ij}\sim U(1,1000000),\ i=1,...,m,\ j=1,...,n$

Class 3
Πίνακες ακεραίων ώστε $1\le c_{ij}\le ij,\ i=1,...,m,\ j=1,...,n$.

6.3.2 Οι κλάσεις στο πρόβλημα Αντιστοίχησης

Οι $n\times n$ πίνακες δεδομένων c_{ij} που χρησιμοποιήσαμε για τη διεξαγωγή της υπολογιστικής μελέτης ανήκουν στις παρακάτω κλάσεις:

Class 1
Ασσυμετρικοί πίνακες πραγματικών αριθμών ομοιόμορφα κατανεμημένων στο διάστημα $[1,100)$. Δηλαδή $c_{ij}\sim U(1,100),\ i=1,...,n,\ j=1,...,n$.

Class 2
Ασσυμετρικοί πίνακες πραγματικών αριθμών ομοιόμορφα κατανεμημένων στο διάστημα $[1,1000000)$. Δηλαδή $c_{ij}\sim U(1,1000000),\ i=1,...,n,\ j=1,...,n$

Class 3
Πίνακες ακεραίων ώστε $1\le c_{ij}\le ij,\ i=1,...,n,\ j=1,...,n$.

Class 4

Συμμετρικοί πίνακες πραγματικών αριθμών ομοιόμορφα κατανεμημένων στο διάστημα $[1,100)$. Δηλαδή $c_{ij} \sim U(1,1000000) \wedge c_{ij} = c_{ji},\ i = 1,...,n,\ j \geq i$

Class 5
Ασσυμετρικοί μεροληπτικοί (*biased*) πίνακες πραγματικών αριθμών. Οι πίνακες αυτοί κατασκευάζονται ως εξής. Επιλέγοντας τυχαία 5 ακεραίους $l(1)$, $l(2)$, $l(3)$, $l(4)$, $l(5)$ μεταξύ 1 και n κατανέμουμε ομοιόμορφα στο διάστημα $[1,10)$ τα στοιχεία των γραμμών $l(1)$, $l(2)$, $l(3)$, $l(4)$, $l(5)$, ενώ τα στοιχεία των εναπομεινάντων $n-5$ γραμμών κατανέμονται ομοιόμορφα στο $[1,1000)$.

Class 6
Η κλάση αυτή ταυτίζεται με τα αντίστοιχα αρνητικά στοιχεία της κλάσης 1. Δηλαδή $c_{ij} \sim U(-100,-1),\ i = 1,...,n,\ j = 1,...,n$.

Class 7
Ασσυμετρικοί πίνακες πραγματικών αριθμών ομοιόμορφα κατανεμημένων στο διάστημα $[-100,100)$ ώστε 50% των στοιχείων ($n^2/2$) να είναι αρνητικά και το υπόλοιπο 50% να είναι θετικά.

6.4 ΥΠΟΛΟΓΙΣΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ

Παρακάτω παρουσιάζουμε τους συγκεντρωτικούς πίνακες των αποτελεσμάτων για τα προβλήματα Μεταφοράς και Αντιστοίχησης καθώς και τα αντίστοιχα διαγράμματα. Σε κάθε πίνακα θα αναγράφεται ο CPU χρόνος και ο αριθμός των επαναλήψεων. Τρία είδη πινάκων για κάθε κλάση θα παρουσιάζονται, ένας που θα παρουσιάζει τα πραγματικά αποτελέσματα, ένας που θα δείχνει τη σύγκριση μεταξύ των αλγορίθμων[4], και ένας που δείχνει τις κανονικοποιημένες (*normalized*) τιμές. Ο χρόνος και ο αριθμός επαναλήψεων κάθε αλγορίθμου είναι ο μέσος όρος των αντίστοιχων μεγεθών που προέκυψαν από την εκτέλεση 10 διαφορετικών προβλημάτων της συγκεκριμένης κλάσης και μεγέθους. Η υπολογιστική μελέτη εκτελέσθηκε σε υπολογιστή με επεξεργαστή 800EB MHz Pentium III, RAM 512Mb 133 MHz και λειτουργικό σύστημα Windows 2000 Pro.

6.4.1 Αποτελέσματα στο πρόβλημα Μεταφοράς

Τα μεγέθη των προβλημάτων που χρησιμοποιήθηκαν είναι $n \times 2n$, $n \times n$, $2n \times n$. Για κάθε μία από τις τρεις κλάσεις εκτελέσθηκαν και τα τρία είδη προβλήματων. Τα αποτελέσματα που πήραμε για κάθε κλάση ξεχωριστά -και τα αντίστοιχα διαγράμματα- φαίνονται στα παρακάτω σχήματα. Διαγράμματα θα παρουσιασθούν μόνο για τα κανονικοποιημένα αποτελέσματα.

Στα προβλήματα μεταφοράς $n \times 2n$, $2n \times n$, το $n \in [50,200]$ με βήμα 25. Στα προβλήματα της μορφής $n \times n$, $n \in [50,300]$ με βήμα 25. Σε κάθε κατηγορία τρέξαμε 10 προβλήματα και πήραμε τους μέσους όρους, όπως αναφέρθηκε και παραπάνω. Συνεπώς τρέξαμε $3 \times 11 \times 10 + 6 \times 7 \times 10 = 750$ προβλήματα μεταφοράς.

Στη συνέχεια παρουσιάζονται όλα τα αποτελέσματα που πήραμε για το πρόβλημα μεταφοράς. Θα ακολουθήσουν και τα αποτελέσματα για το πρόβλημα αντιστοίχησης. Θα γίνεται ένας σύντομος σχολιασμός για κάθε περίπτωση που κρίνεται ενδιαφέρουσα. Ο γενικός σχολιασμός των αποτελεσμάτων θα γίνει στο κεφάλαιο των συμπερασμάτων.

---

[4] Αυτό επιτυγχάνεται ως εξής. Αν t_1, t_2 οι χρόνοι εκτέλεσης των δύο αλγορίθμων, παίρνουμε το λόγο $u = t_1 / t_2$. Αν $u > 1$, τότε ο αλγόριθμος με χρόνο t_2 είναι πιο γρήγορος, αλλιώς πιο γρήγορος θα είναι ο άλλος αλγόριθμος.

ΚΛΑΣΗ 1 – nxn

Πίνακας 6.2 - Υπολογιστικά Αποτελέσματα της πρώτης κλάσης δεδομένων, nxn

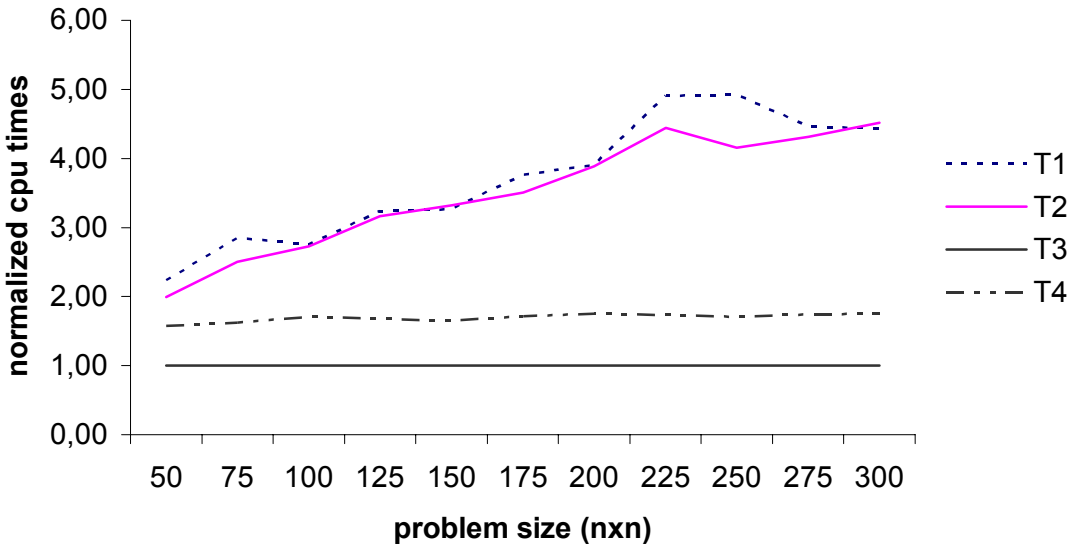
n	T1		T2		T3		T4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	273,3	7,04	152,1	6,28	89,3	3,15	137,0	4,97
75	504,5	26,77	285,0	23,56	151,9	9,41	226,9	15,30
100	752,5	55,32	410,4	54,75	223,2	20,08	319,4	34,33
125	1.035,0	117,71	588,0	114,83	262,7	36,32	392,5	61,03
150	1.354,3	204,76	765,4	208,26	331,0	62,77	483,7	103,84
175	1.694,1	338,58	895,3	315,85	378,3	90,02	558,3	154,18
200	2.007,4	513,05	1.135,5	511,57	457,2	131,53	656,7	230,98
225	2.372,3	907,01	1.427,8	821,73	497,4	184,97	726,6	321,48
250	2.795,0	1.267,27	1.566,6	1.069,54	591,6	257,50	842,4	440,51
275	3.226,5	1.499,80	1.775,5	1.448,67	640,7	336,20	930,1	586,71
300	3.575,0	1.957,07	2.050,1	1.994,09	724,5	441,18	1.038,7	771,86

Πίνακας 6.3 - Σύγκριση των αλγορίθμων για την πρώτη κλάση δεδομένων, nxn

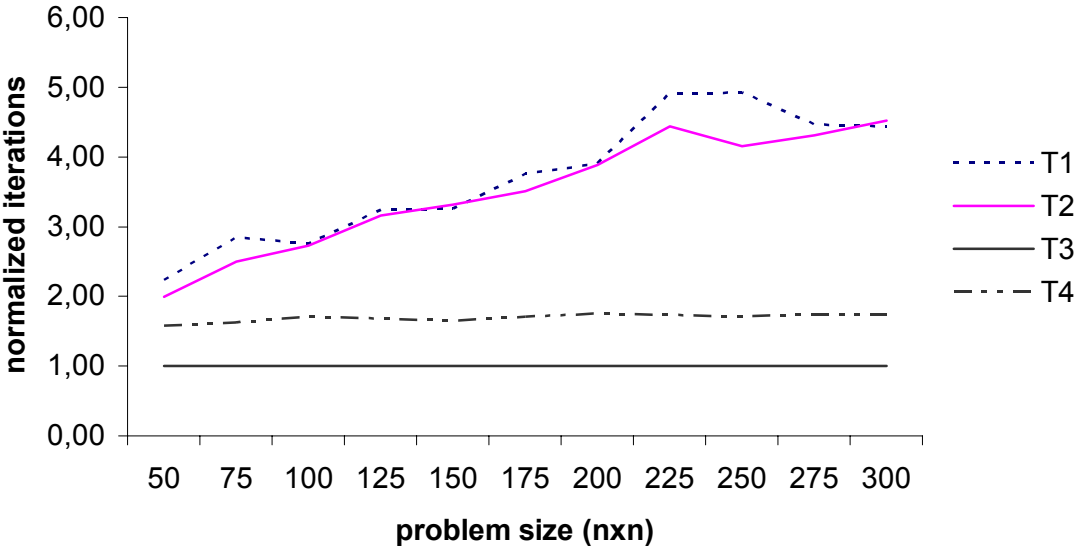
	T1/T2		T1/T3		T1/T4		T2/T3		T2/T4		T4/T3	
n	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	1,797	1,121	3,060	2,235	1,995	1,416	1,703	1,994	1,110	1,416	1,534	1,578
75	1,770	1,136	3,321	2,845	2,223	1,750	1,876	2,504	1,256	1,750	1,494	1,626
100	1,834	1,010	3,371	2,755	2,356	1,611	1,839	2,727	1,285	1,611	1,431	1,710
125	1,760	1,025	3,940	3,241	2,637	1,929	2,238	3,162	1,498	1,929	1,494	1,680
150	1,769	0,983	4,092	3,262	2,800	1,972	2,312	3,318	1,582	1,972	1,461	1,654
175	1,892	1,072	4,478	3,761	3,034	2,196	2,367	3,509	1,604	2,196	1,476	1,713
200	1,768	1,003	4,391	3,901	3,057	2,221	2,484	3,889	1,729	2,221	1,436	1,756
225	1,662	1,104	4,769	4,904	3,265	2,821	2,871	4,443	1,965	2,821	1,461	1,738
250	1,784	1,185	4,724	4,921	3,318	2,877	2,648	4,154	1,860	2,877	1,424	1,711
275	1,817	1,035	5,036	4,461	3,469	2,556	2,771	4,309	1,909	2,556	1,452	1,745
300	1,744	0,981	4,934	4,436	3,442	2,536	2,830	4,520	1,974	2,536	1,434	1,750

Πίνακας 6.4 - Κανονικοποιημένα αποτελέσματα για την πρώτη κλάση, nxn

n	T1		T2		T3		T4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	3,06	2,23	1,70	1,99	1,00	1,00	1,53	1,58
75	3,32	2,84	1,88	2,50	1,00	1,00	1,49	1,63
100	3,37	2,75	1,84	2,73	1,00	1,00	1,43	1,71
125	3,94	3,24	2,24	3,16	1,00	1,00	1,49	1,68
150	4,09	3,26	2,31	3,32	1,00	1,00	1,46	1,65
175	4,48	3,76	2,37	3,51	1,00	1,00	1,48	1,71
200	4,39	3,90	2,48	3,89	1,00	1,00	1,44	1,76
225	4,77	4,90	2,87	4,44	1,00	1,00	1,46	1,74
250	4,72	4,92	2,65	4,15	1,00	1,00	1,42	1,71
275	5,04	4,46	2,77	4,31	1,00	1,00	1,45	1,75
300	4,93	4,44	2,83	4,52	1,00	1,00	1,43	1,75



Εικόνα 6.0 – Κλάση 1:Το διάγραμμα των κανονικοποιημένων χρόνων, nxn



Εικόνα 6.1– Κλάση 1 (nxn): Το διάγραμμα των κανονικοποιημένων επαναλήψεων, nxn

Στα παραπάνω διαγράμματα, παρατηρούμε ότι και στις επαναλήψεις αλλά και στο χρόνο CPU, ο αλγόριθμος T3 είναι σαφώς ανώτερος. Συγκεκριμένα, για μεγάλα μεγέθη προβλημάτων, η διαφορά φτάνει έναν συντελεστή μεγέθους 4. Η κατάταξη που παρατηρούμε είναι η εξής: T3, T4, και οι αλγόριθμοι T2, T1 συναγωνίζονται για την προτελευταία θέση.

ΚΛΑΣΗ 1 – nx2n

Πίνακας 6.5 - Υπολογιστικά Αποτελέσματα της πρώτης κλάσης δεδομένων, nx2n

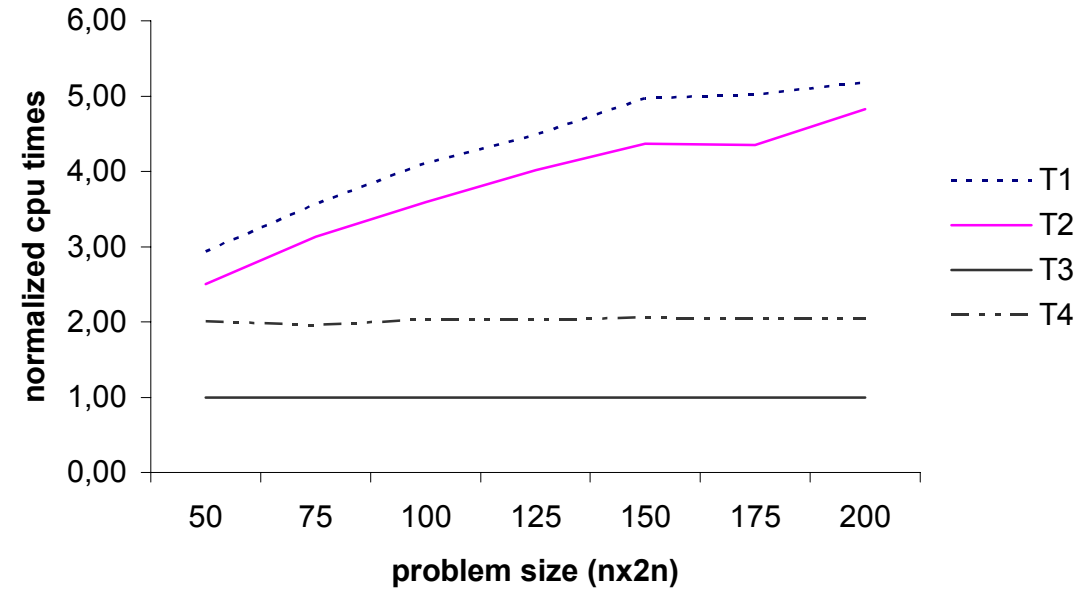
nx2n	T1		T2		T3		T4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	469,2	23,14	256,3	19,75	132,6	7,88	235,8	15,85
75	861,2	85,59	481,8	75,21	212,8	24,02	358,8	47,01
100	1.284,5	215,66	727,6	188,77	293,2	52,51	504,6	106,90
125	1.742,7	436,04	1.003,6	390,68	381,3	97,23	637,5	197,06
150	2.303,8	809,43	1.325,8	710,91	450,1	162,74	759,1	334,63
175	2.875,0	1.349,90	1.618,3	1.167,29	581,1	268,61	951,5	548,82
200	3.407,8	2.032,94	2.039,3	1.891,30	669,1	392,03	1.097,0	801,55

Πίνακας 6.6 - Σύγκριση των αλγορίθμων για την πρώτη κλάση δεδομένων, nx2n

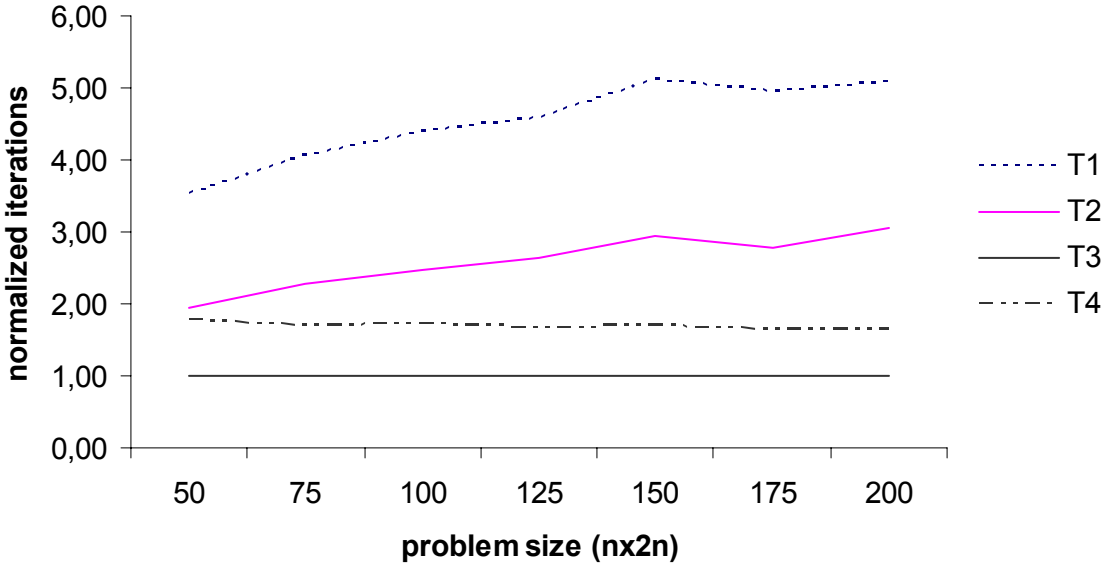
	T1/T2		T1/T3		T1/T4		T2/T3		T2/T4		T4/T3	
nx2n	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	1,831	1,172	3,538	2,937	1,990	1,460	1,933	2,506	1,087	1,460	1,778	2,011
75	1,787	1,138	4,047	3,563	2,400	1,821	2,264	3,131	1,343	1,821	1,686	1,957
100	1,765	1,142	4,381	4,107	2,546	2,017	2,482	3,595	1,442	2,017	1,721	2,036
125	1,736	1,116	4,570	4,485	2,734	2,213	2,632	4,018	1,574	2,213	1,672	2,027
150	1,738	1,139	5,118	4,974	3,035	2,419	2,946	4,368	1,747	2,419	1,687	2,056
175	1,777	1,156	4,948	5,026	3,022	2,460	2,785	4,346	1,701	2,460	1,637	2,043
200	1,671	1,075	5,093	5,186	3,106	2,536	3,048	4,824	1,859	2,536	1,640	2,045

Πίνακας 6.7 - Κανονικοποιημένα αποτελέσματα για την πρώτη κλάση, nx2n

nx2n	T1		T2		T3		T4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	3,54	2,94	1,93	2,51	1,00	1,00	1,78	2,01
75	4,05	3,56	2,26	3,13	1,00	1,00	1,69	1,96
100	4,38	4,11	2,48	3,59	1,00	1,00	1,72	2,04
125	4,57	4,48	2,63	4,02	1,00	1,00	1,67	2,03
150	5,12	4,97	2,95	4,37	1,00	1,00	1,69	2,06
175	4,95	5,03	2,78	4,35	1,00	1,00	1,64	2,04
200	5,09	5,19	3,05	4,82	1,00	1,00	1,64	2,04



Εικόνα 6.2 – Κλάση 1 :Το διάγραμμα των κανονικοποιημένων χρόνων, nx2n



Εικόνα 6.3 – Κλάση 1 (nx2n):Το διάγραμμα των κανονικοποιημένων επαναλήψεων, nx2n

Εδώ παρατηρούμε ότι η υπεροχή των αλγορίθμων εξωτερικών σημείων είναι εμφανής, ιδιαίτερα στον αριθμό των επαναλήψεων. Φαίνεται ξεκάθαρα ότι ο αλγόριθμος simplex είναι λιγότερο αποδοτικός. Πρώτος παραμένει σταθερά ο αλγόριθμος T3.

ΚΛΑΣΗ 1 – 2nxn

Πίνακας 6.8 - Υπολογιστικά Αποτελέσματα της πρώτης κλάσης δεδομένων, 2nxn

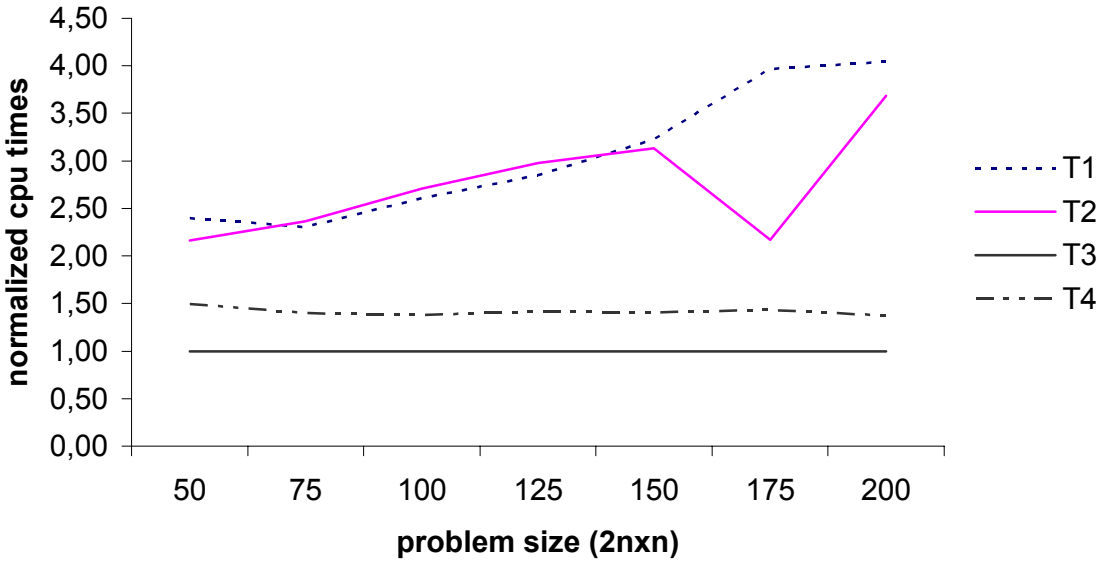
2nxn	T1		T2		T3		T4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	476,4	23,08	264,0	20,80	171,4	9,62	221,8	14,39
75	833,3	71,80	466,4	73,73	288,0	31,18	357,8	43,71
100	1.257,6	178,01	700,9	184,83	383,5	68,20	480,8	94,01
125	1.749,9	371,35	989,9	388,82	505,8	130,46	644,6	184,40
150	2.307,1	686,71	1.208,2	665,98	597,4	212,60	743,4	299,14
175	2.832,9	1.326,01	1.219,2	725,30	725,3	334,15	906,6	479,68
200	3.443,1	2.071,15	1.982,2	1.888,26	884,3	512,54	1.079,7	703,67

Πίνακας 6.9 - Σύγκριση των αλγορίθμων για την πρώτη κλάση δεδομένων, 2nxn

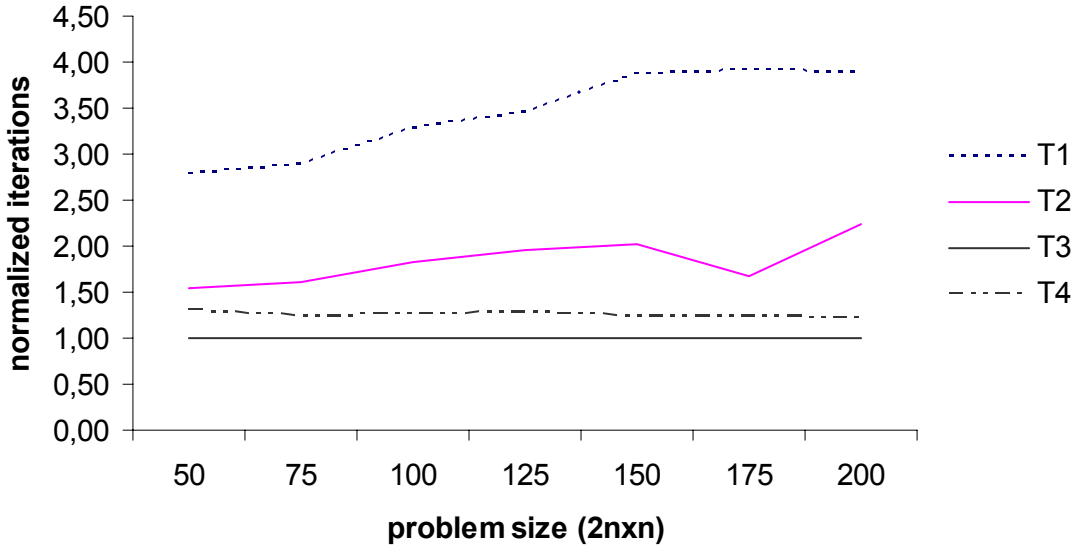
	T1/T2		T1/T3		T1/T4		T2/T3		T2/T4		T4/T3	
2nxn	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	1,805	1,110	2,779	2,399	2,148	1,604	1,540	2,162	1,190	1,604	1,294	1,496
75	1,787	0,974	2,893	2,303	2,329	1,643	1,619	2,365	1,304	1,643	1,242	1,402
100	1,794	0,963	3,279	2,610	2,616	1,894	1,828	2,710	1,458	1,894	1,254	1,378
125	1,768	0,955	3,460	2,846	2,715	2,014	1,957	2,980	1,536	2,014	1,274	1,413
150	1,910	1,031	3,862	3,230	3,103	2,296	2,022	3,133	1,625	2,296	1,244	1,407
175	2,324	1,828	3,906	3,968	3,125	2,764	1,681	2,171	1,345	2,764	1,250	1,436
200	1,737	1,097	3,894	4,041	3,189	2,943	2,242	3,684	1,836	2,943	1,221	1,373

Πίνακας 6.10 - Κανονικοποιημένα αποτελέσματα για την πρώτη κλάση, 2nxn

2nxn	T1		T2		T3		T4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	2,78	2,40	1,54	2,16	1,00	1,00	1,29	1,50
75	2,89	2,30	1,62	2,36	1,00	1,00	1,24	1,40
100	3,28	2,61	1,83	2,71	1,00	1,00	1,25	1,38
125	3,46	2,85	1,96	2,98	1,00	1,00	1,27	1,41
150	3,86	3,23	2,02	3,13	1,00	1,00	1,24	1,41
175	3,91	3,97	1,68	2,17	1,00	1,00	1,25	1,44
200	3,89	4,04	2,24	3,68	1,00	1,00	1,22	1,37



Εικόνα 6.4 – Κλάση 1 :Το διάγραμμα των κανονικοποιημένων χρόνων, 2nxn



Εικόνα 6.5 – Κλάση 1 (2nxn):Το διάγραμμα των κανονικοποιημένων επαναλήψεων, 2nxn

Στα διαγράμματα που παρουσιάσαμε φαίνεται και πάλι η υπεροχή του αλγορίθμου T3. Αξιοσημείωτο είναι το γεγονός ότι ο αλγόριθμος T2 για $n \simeq 180$ βελτιώνει σημαντικά και το χρόνο εκτέλεσης για να τον αυξήσει στη συνέχεια για μεγαλύτερα μεγέθη.

ΚΛΑΣΗ 2 – nxn

Πίνακας 6.11 - Υπολογιστικά Αποτελέσματα της δεύτερης κλάσης δεδομένων, nxn

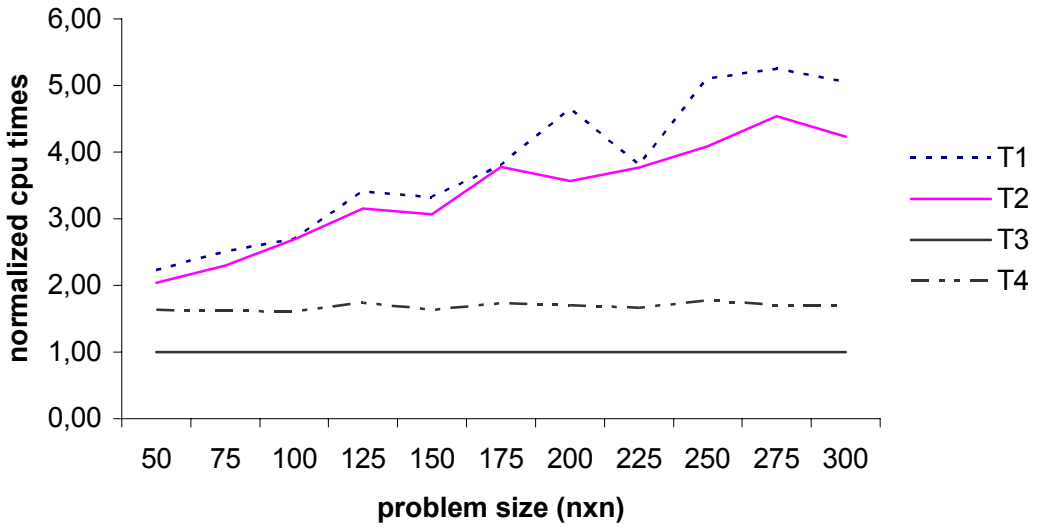
n	T1		T2		T3		T4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	261,0	6,79	155,9	6,20	89,2	3,04	140,1	4,97
75	506,7	23,61	273,0	21,56	151,7	9,39	224,6	15,16
100	757,6	55,25	426,3	55,04	214,8	20,42	312,1	32,78
125	1.034,2	114,99	559,9	106,29	252,1	33,71	377,1	58,72
150	1.340,2	204,06	717,6	188,16	325,5	61,27	474,6	99,89
175	1.660,0	332,72	921,8	330,19	364,5	87,41	540,5	151,47
200	2.059,8	628,44	1.083,4	481,85	449,6	135,21	655,3	230,00
225	2.451,7	764,47	1.359,4	757,38	541,4	200,91	772,2	333,37
250	2.811,0	1.294,36	1.516,2	1.035,18	572,8	253,61	848,3	450,92
275	3.152,3	1.731,11	1.821,7	1.495,49	625,9	329,85	893,2	562,84
300	3.554,2	2.270,05	1.969,5	1.897,86	722,6	448,69	1.031,2	763,10

Πίνακας 6.12 - Σύγκριση των αλγορίθμων για τη δεύτερη κλάση δεδομένων, nxn

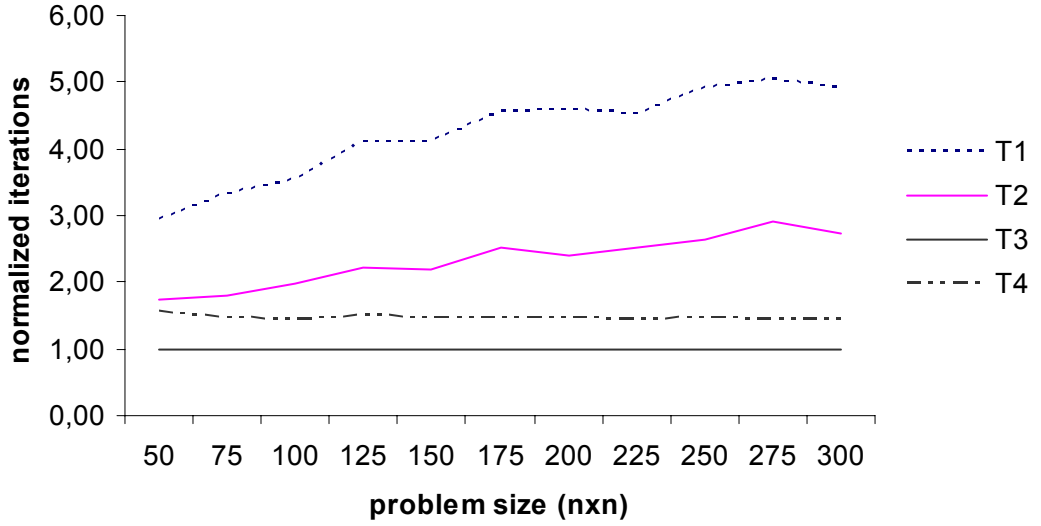
	T1/T2		T1/T3		T1/T4		T2/T3		T2/T4		T4/T3	
n	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	1,674	1,095	2,926	2,234	1,863	1,366	1,748	2,039	1,113	1,366	1,571	1,635
75	1,856	1,095	3,340	2,514	2,256	1,557	1,800	2,296	1,215	1,557	1,481	1,614
100	1,777	1,004	3,527	2,706	2,427	1,685	1,985	2,695	1,366	1,685	1,453	1,605
125	1,847	1,082	4,102	3,411	2,743	1,958	2,221	3,153	1,485	1,958	1,496	1,742
150	1,868	1,085	4,117	3,331	2,824	2,043	2,205	3,071	1,512	2,043	1,458	1,630
175	1,801	1,008	4,554	3,806	3,071	2,197	2,529	3,777	1,705	2,197	1,483	1,733
200	1,901	1,304	4,581	4,648	3,143	2,732	2,410	3,564	1,653	2,732	1,458	1,701
225	1,804	1,009	4,528	3,805	3,175	2,293	2,511	3,770	1,760	2,293	1,426	1,659
250	1,854	1,250	4,907	5,104	3,314	2,870	2,647	4,082	1,787	2,870	1,481	1,778
275	1,730	1,158	5,036	5,248	3,529	3,076	2,911	4,534	2,040	3,076	1,427	1,706
300	1,805	1,196	4,919	5,059	3,447	2,975	2,726	4,230	1,910	2,975	1,427	1,701

Πίνακας 6.13 - Κανονικοποιημένα αποτελέσματα για τη δεύτερη κλάση, nxn

n	T1		T2		T3		T4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	2,93	2,23	1,75	2,04	1,00	1,00	1,57	1,63
75	3,34	2,51	1,80	2,30	1,00	1,00	1,48	1,61
100	3,53	2,71	1,98	2,70	1,00	1,00	1,45	1,61
125	4,10	3,41	2,22	3,15	1,00	1,00	1,50	1,74
150	4,12	3,33	2,20	3,07	1,00	1,00	1,46	1,63
175	4,55	3,81	2,53	3,78	1,00	1,00	1,48	1,73
200	4,58	4,65	2,41	3,56	1,00	1,00	1,46	1,70
225	4,53	3,81	2,51	3,77	1,00	1,00	1,43	1,66
250	4,91	5,10	2,65	4,08	1,00	1,00	1,48	1,78
275	5,04	5,25	2,91	4,53	1,00	1,00	1,43	1,71
300	4,92	5,06	2,73	4,23	1,00	1,00	1,43	1,70



Εικόνα 6.6 – Κλάση 2 :Το διάγραμμα των κανονικοποιημένων χρόνων, nxn



Εικόνα 6.7 – Κλάση 2 :Το διάγραμμα των κανονικοποιημένων επαναλήψεων, nxn

Παρατηρούμε ότι η εικόνα κατάταξης των αλγορίθμων παραμένει η ίδια. Στους χρόνους εκτέλεσης, τις τελευταίες θέσεις καταλαμβάνουν οι αλγόριθμοι T2,T1.

ΚΛΑΣΗ 2 – nx2n

Πίνακας 6.14 - Υπολογιστικά Αποτελέσματα της δεύτερης κλάσης δεδομένων, nx2n

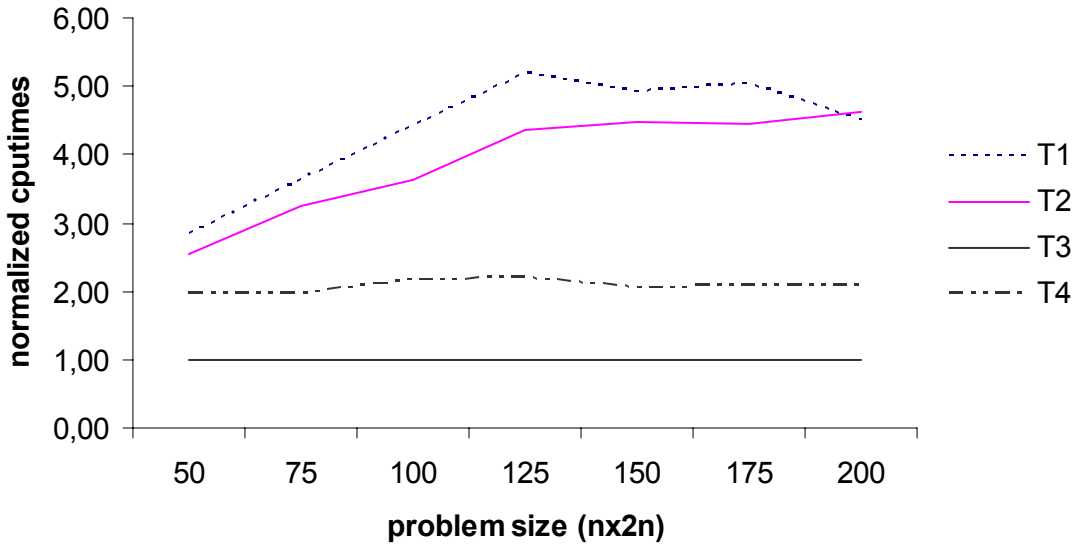
nx2n	T1		T2		T3		T4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	451,3	22,29	261,5	20,02	135,4	7,82	232,4	15,31
75	823,9	86,96	490,8	77,78	222,1	24,01	362,4	47,14
100	1.279,4	224,19	702,0	184,78	294,8	50,83	509,5	110,23
125	1.750,8	467,18	989,1	395,21	363,8	90,34	617,8	199,27
150	2.278,1	804,31	1.329,2	730,33	459,2	163,15	770,0	335,47
175	2.857,1	1.300,25	1.573,4	1.151,28	570,3	258,46	942,6	538,71
200	3.486,7	1.783,20	1.958,1	1.831,08	673,4	395,62	1.095,8	818,45

Πίνακας 6.15 - Σύγκριση των αλγορίθμων για τη δεύτερη κλάση δεδομένων, nx2n

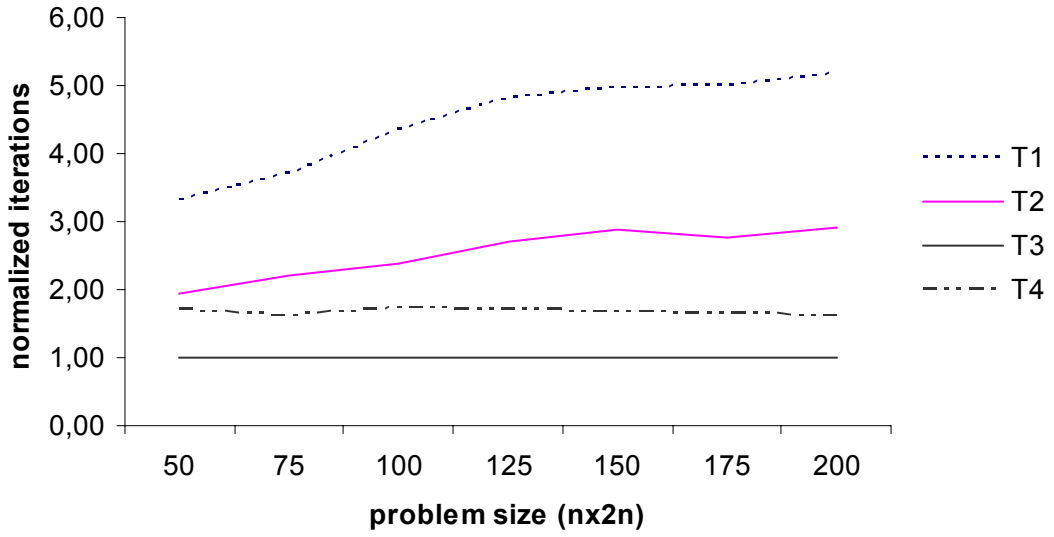
	T1/T2		T1/T3		T1/T4		T2/T3		T2/T4		T4/T3	
nx2n	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	1,726	1,113	3,333	2,850	1,942	1,456	1,931	2,560	1,125	1,456	1,716	1,958
75	1,679	1,118	3,710	3,622	2,273	1,845	2,210	3,239	1,354	1,845	1,632	1,963
100	1,823	1,213	4,340	4,411	2,511	2,034	2,381	3,635	1,378	2,034	1,728	2,169
125	1,770	1,182	4,813	5,171	2,834	2,344	2,719	4,375	1,601	2,344	1,698	2,206
150	1,714	1,101	4,961	4,930	2,959	2,398	2,895	4,476	1,726	2,398	1,677	2,056
175	1,816	1,129	5,010	5,031	3,031	2,414	2,759	4,454	1,669	2,414	1,653	2,084
200	1,781	0,974	5,178	4,507	3,182	2,179	2,908	4,628	1,787	2,179	1,627	2,069

Πίνακας 6.16 - Κανονικοποιημένα αποτελέσματα για τη δεύτερη κλάση, nx2n

nx2n	T1		T2		T3		T4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	3,33	2,85	1,93	2,56	1,00	1,00	1,72	1,96
75	3,71	3,62	2,21	3,24	1,00	1,00	1,63	1,96
100	4,34	4,41	2,38	3,64	1,00	1,00	1,73	2,17
125	4,81	5,17	2,72	4,37	1,00	1,00	1,70	2,21
150	4,96	4,93	2,89	4,48	1,00	1,00	1,68	2,06
175	5,01	5,03	2,76	4,45	1,00	1,00	1,65	2,08
200	5,18	4,51	2,91	4,63	1,00	1,00	1,63	2,07



Εικόνα 6.8 – Κλάση 2 :Το διάγραμμα των κανονικοποιημένων χρόνων, nx2n



Εικόνα 6.9 – Κλάση 2 :Το διάγραμμα των κανονικοποιημένων επαναλήψεων, nx2n

ΚΛΑΣΗ 2 – 2nxn

Πίνακας 6.17 - Υπολογιστικά Αποτελέσματα της δεύτερης κλάσης δεδομένων, 2nxn

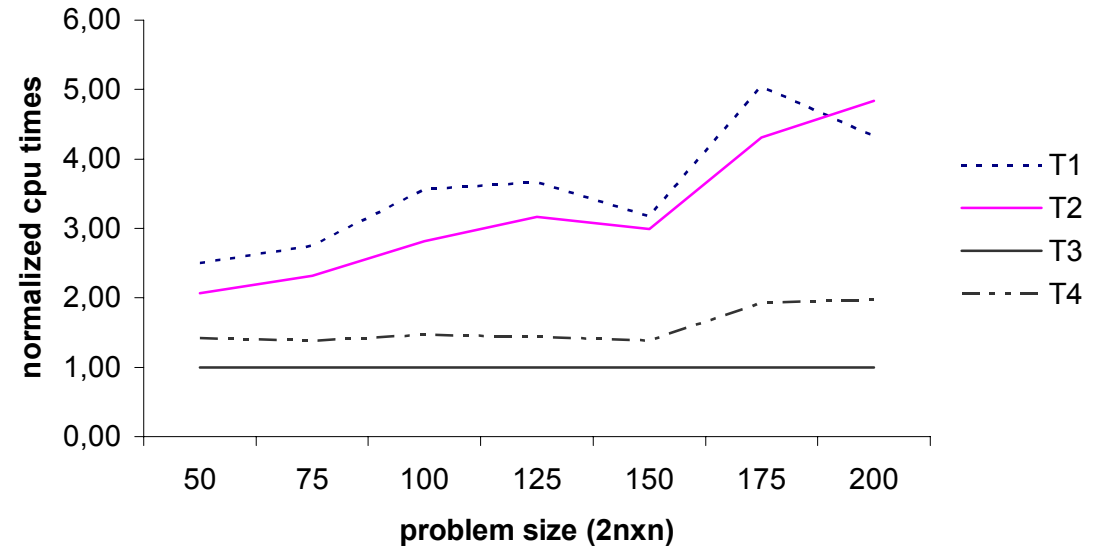
2nxn	T1		T2		T3		T4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	469,2	24,63	263,2	20,40	172,5	9,85	225,5	13,97
75	845,4	84,51	450,2	71,32	285,6	30,73	353,5	42,48
100	1.296,9	234,03	694,5	184,52	378,1	65,61	484,0	96,19
125	1.769,0	445,37	962,7	385,41	480,1	121,65	603,1	175,56
150	2.325,6	712,05	1.210,9	671,84	589,7	224,49	733,2	311,18
175	2.890,8	1.412,21	1.647,3	1.208,47	595,2	280,12	941,0	541,70
200	3.441,2	1.737,13	2.050,8	1.940,25	673,8	401,21	1.085,3	793,20

Πίνακας 6.18 - Σύγκριση των αλγορίθμων για τη δεύτερη κλάση δεδομένων, 2nxn

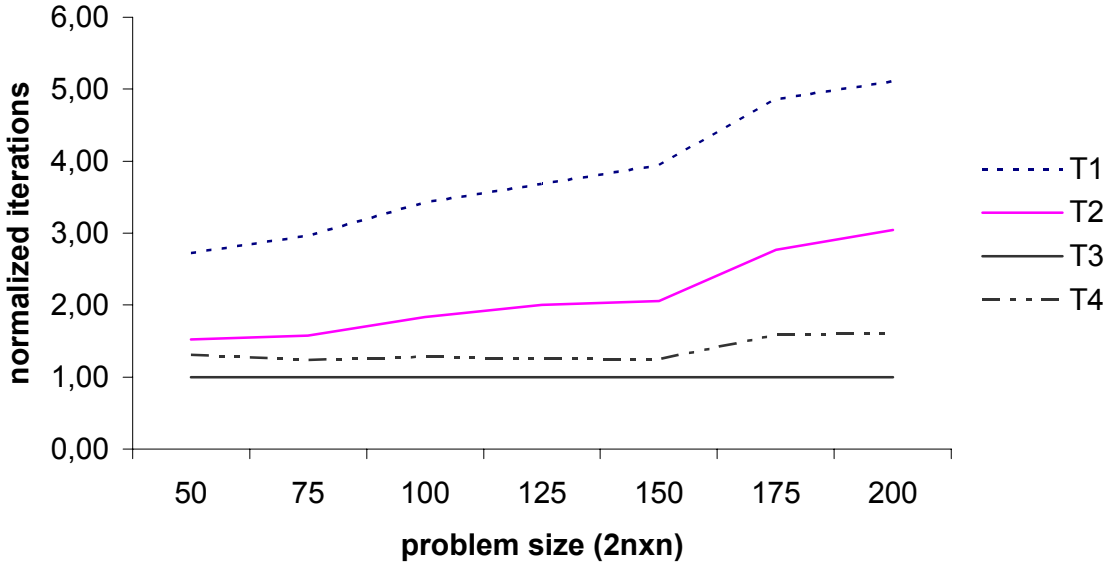
	T1/T2		T1/T3		T1/T4		T2/T3		T2/T4		T4/T3	
2nxn	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	1,783	1,207	2,720	2,499	2,081	1,763	1,526	2,070	1,167	1,763	1,307	1,417
75	1,878	1,185	2,960	2,750	2,392	1,989	1,576	2,321	1,274	1,989	1,238	1,382
100	1,867	1,268	3,430	3,567	2,680	2,433	1,837	2,812	1,435	2,433	1,280	1,466
125	1,838	1,156	3,685	3,661	2,933	2,537	2,005	3,168	1,596	2,537	1,256	1,443
150	1,921	1,060	3,944	3,172	3,172	2,288	2,053	2,993	1,652	2,288	1,243	1,386
175	1,755	1,169	4,857	5,041	3,072	2,607	2,768	4,314	1,751	2,607	1,581	1,934
200	1,678	0,895	5,107	4,330	3,171	2,190	3,044	4,836	1,890	2,190	1,611	1,977

Πίνακας 6.19 - Κανονικοποιημένα αποτελέσματα για τη δεύτερη κλάση, 2nxn

2nxn	T1		T2		T3		T4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	2,72	2,50	1,53	2,07	1,00	1,00	1,31	1,42
75	2,96	2,75	1,58	2,32	1,00	1,00	1,24	1,38
100	3,43	3,57	1,84	2,81	1,00	1,00	1,28	1,47
125	3,68	3,66	2,01	3,17	1,00	1,00	1,26	1,44
150	3,94	3,17	2,05	2,99	1,00	1,00	1,24	1,39
175	4,86	5,04	2,77	4,31	1,00	1,00	1,58	1,93
200	5,11	4,33	3,04	4,84	1,00	1,00	1,61	1,98



Εικόνα 6.10 – Κλάση 2 :Το διάγραμμα των κανονικοποιημένων χρόνων, 2nxn



Εικόνα 6.11 – Κλάση 2 :Το διάγραμμα των κανονικοποιημένων επαναλήψεων, 2nxn

Αυτό που πρέπει να πούμε σε αυτήν την κλάση δεδομένων είναι το γεγονός ότι για μεγάλα μεγέθη προβλήματος, φαίνεται ότι ο αλγόριθμος simplex είναι πιο αποδοτικός από τον αλγόριθμο T2. Αυτό φαίνεται καθαρά στην εικόνα 6.10.

ΚΛΑΣΗ 3 – nxn

Πίνακας 6.20 - Υπολογιστικά Αποτελέσματα της τρίτης κλάσης δεδομένων, nxn

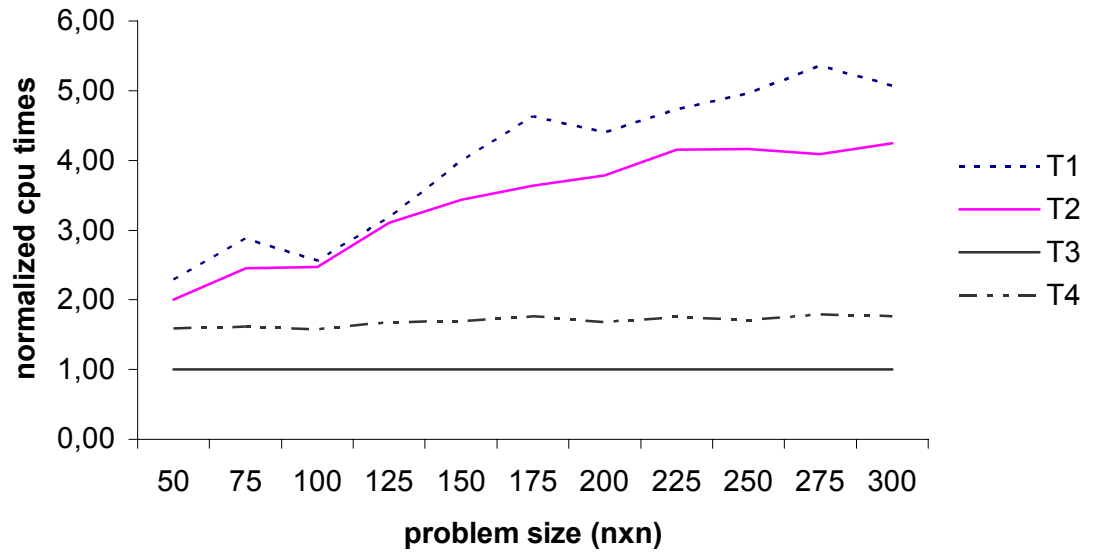
n	T1		T2		T3		T4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	269,5	6,95	151,8	6,08	89,0	3,04	136,6	4,82
75	508,1	26,66	283,9	22,72	151,4	9,26	226,2	14,98
100	759,5	55,17	410,0	53,20	223,8	21,51	320,5	33,93
125	1.032,7	113,70	587,1	111,02	262,2	35,77	391,0	60,22
150	1.359,5	241,72	764,8	208,35	329,8	60,68	483,1	102,79
175	1.712,1	408,23	896,7	320,65	377,2	88,20	558,5	155,79
200	2.000,2	605,07	1.135,7	520,46	458,1	137,50	655,8	231,44
225	2.325,9	871,59	1.318,4	764,67	519,7	184,25	749,1	324,08
250	2.789,2	1.274,09	1.563,7	1.068,19	591,1	256,80	842,3	439,00
275	3.165,5	1.753,92	1.616,0	1.338,75	621,5	327,69	921,0	585,70
300	3.594,2	2.231,47	1.938,4	1.867,91	718,8	440,35	1.036,7	778,83

Πίνακας 6.21 - Σύγκριση των αλγορίθμων για την τρίτη κλάση δεδομένων, nxn

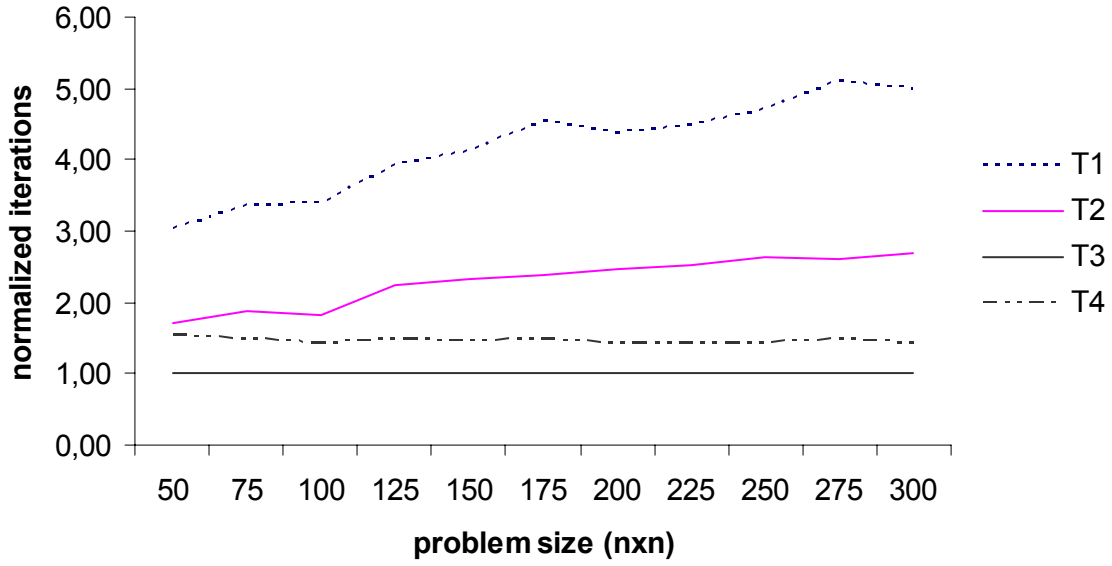
	T1/T2		T1/T3		T1/T4		T2/T3		T2/T4		T4/T3	
n	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	1,775	1,143	3,028	2,286	1,973	1,442	1,706	2,000	1,111	1,442	1,535	1,586
75	1,790	1,173	3,356	2,879	2,246	1,780	1,875	2,454	1,255	1,780	1,494	1,618
100	1,852	1,037	3,394	2,565	2,370	1,626	1,832	2,473	1,279	1,626	1,432	1,577
125	1,759	1,024	3,939	3,179	2,641	1,888	2,239	3,104	1,502	1,888	1,491	1,684
150	1,778	1,160	4,122	3,984	2,814	2,352	2,319	3,434	1,583	2,352	1,465	1,694
175	1,909	1,273	4,539	4,628	3,066	2,620	2,377	3,635	1,606	2,620	1,481	1,766
200	1,761	1,163	4,366	4,401	3,050	2,614	2,479	3,785	1,732	2,614	1,432	1,683
225	1,764	1,140	4,475	4,730	3,105	2,689	2,537	4,150	1,760	2,689	1,441	1,759
250	1,784	1,193	4,719	4,961	3,311	2,902	2,645	4,160	1,856	2,902	1,425	1,710
275	1,959	1,310	5,093	5,352	3,437	2,995	2,600	4,085	1,755	2,995	1,482	1,787
300	1,854	1,195	5,000	5,067	3,467	2,865	2,697	4,242	1,870	2,865	1,442	1,769

Πίνακας 6.22 - Κανονικοποιημένα αποτελέσματα για την τρίτη κλάση δεδομένων, nxn

n	T1		T2		T3		T4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	3,03	2,29	1,71	2,00	1,00	1,00	1,53	1,59
75	3,36	2,88	1,88	2,45	1,00	1,00	1,49	1,62
100	3,39	2,56	1,83	2,47	1,00	1,00	1,43	1,58
125	3,94	3,18	2,24	3,10	1,00	1,00	1,49	1,68
150	4,12	3,98	2,32	3,43	1,00	1,00	1,46	1,69
175	4,54	4,63	2,38	3,64	1,00	1,00	1,48	1,77
200	4,37	4,40	2,48	3,79	1,00	1,00	1,43	1,68
225	4,48	4,73	2,54	4,15	1,00	1,00	1,44	1,76
250	4,72	4,96	2,65	4,16	1,00	1,00	1,42	1,71
275	5,09	5,35	2,60	4,09	1,00	1,00	1,48	1,79
300	5,00	5,07	2,70	4,24	1,00	1,00	1,44	1,77



Εικόνα 6.12 – Κλάση 3:Το διάγραμμα των κανονικοποιημένων χρόνων, nxn



Εικόνα 6.13 – Κλάση 3:Το διάγραμμα των κανονικοποιημένων επαναλήψεων, nxn

ΚΛΑΣΗ 3 – nx2n

Πίνακας 6.23 - Υπολογιστικά Αποτελέσματα της τρίτης κλάσης δεδομένων, nx2n

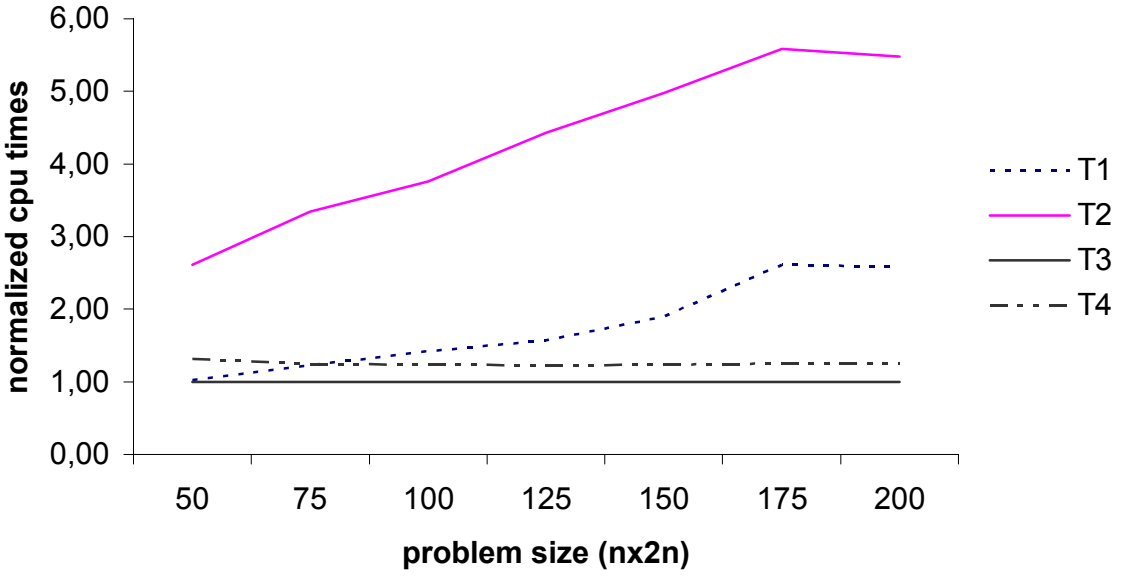
nx2n	T1		T2		T3		T4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	256,1	10,38	361,2	26,39	218,9	10,11	267,2	13,30
75	514,5	37,43	676,9	101,73	339,4	30,46	412,7	38,02
100	818,7	96,49	999,9	255,03	475,0	67,82	572,3	83,51
125	1.090,4	196,85	1.442,5	556,03	613,3	125,54	735,5	154,34
150	1.542,5	387,42	1.857,5	1.016,40	732,0	204,27	877,4	253,09
175	1.856,3	821,34	2.518,9	1.759,50	857,6	314,91	1.072,5	394,49
200	2.309,6	1.209,96	3.028,5	2.561,97	1.037,0	467,33	1.233,0	584,77

Πίνακας 6.24 - Σύγκριση των αλγορίθμων για την τρίτη κλάση δεδομένων, nx2n

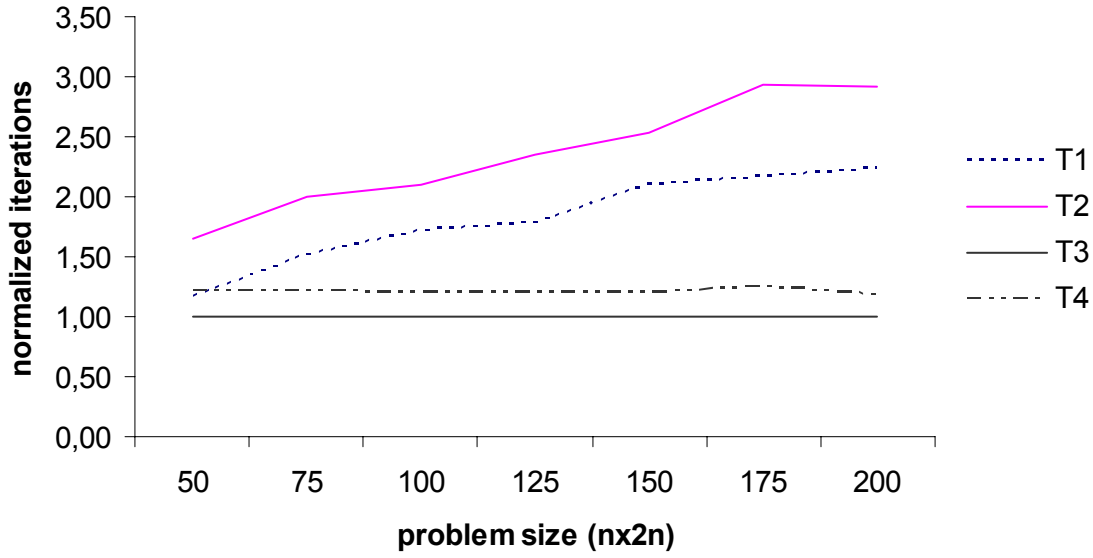
	T1/T2		T1/T3		T1/T4		T2/T3		T2/T4		T4/T3	
nx2n	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	0,709	0,393	1,170	1,027	0,958	0,780	1,650	2,610	1,352	0,780	1,221	1,316
75	0,760	0,368	1,516	1,229	1,247	0,984	1,994	3,340	1,640	0,984	1,216	1,248
100	0,819	0,378	1,724	1,423	1,431	1,155	2,105	3,760	1,747	1,155	1,205	1,231
125	0,756	0,354	1,778	1,568	1,483	1,275	2,352	4,429	1,961	1,275	1,199	1,229
150	0,830	0,381	2,107	1,897	1,758	1,531	2,538	4,976	2,117	1,531	1,199	1,239
175	0,737	0,467	2,165	2,608	1,731	2,082	2,937	5,587	2,349	2,082	1,251	1,253
200	0,763	0,472	2,227	2,589	1,873	2,069	2,920	5,482	2,456	2,069	1,189	1,251

Πίνακας 6.25 - Κανονικοποιημένα αποτελέσματα για την τρίτη κλάση δεδομένων, nx2n

nx2n	T1		T2		T3		T4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	1,17	1,03	1,65	2,61	1,00	1,00	1,22	1,32
75	1,52	1,23	1,99	3,34	1,00	1,00	1,22	1,25
100	1,72	1,42	2,11	3,76	1,00	1,00	1,20	1,23
125	1,78	1,57	2,35	4,43	1,00	1,00	1,20	1,23
150	2,11	1,90	2,54	4,98	1,00	1,00	1,20	1,24
175	2,16	2,61	2,94	5,59	1,00	1,00	1,25	1,25
200	2,23	2,59	2,92	5,48	1,00	1,00	1,19	1,25



Εικόνα 6.14 – Κλάση 3:Το διάγραμμα των κανονικοποιημένων χρόνων, nx2n



Εικόνα 6.15 – Κλάση 3:Το διάγραμμα των κανονικοποιημένων επαναλήψεων, nx2n

Βασική παρατήρηση που προκύπτει από τα παραπάνω διαγράμματα είναι η ανατροπή στην τελευταία θέση των αλγορίθμων. Αυτή η κλάση δεδομένων φαίνεται ότι ευνοεί τον αλγόριθμο simplex (T1) έναντι του αλγορίθμου T2. Καλύτερος παραμένει ο αλγόριθμος T3 και σε επαναλήψεις και σε cpu χρόνο. Επίσης, για μικρά μεγέθη προβλήματος, οι αλγόριθμοι T4, T2, T3 έχουν περίπου ίδια υπολογιστική συμπεριφορά (εικόνα 6.14). Στη συνέχεια παρουσιάζουμε και την εναπομείνασα κλάση δεδομένων για το πρόβλημα μεταφοράς και περνάμε στο πρόβλημα αντιστοίχησης.

ΚΛΑΣΗ 3 – 2nxn

Πίνακας 6.26 - Υπολογιστικά Αποτελέσματα της τρίτης κλάσης δεδομένων, 2nxn

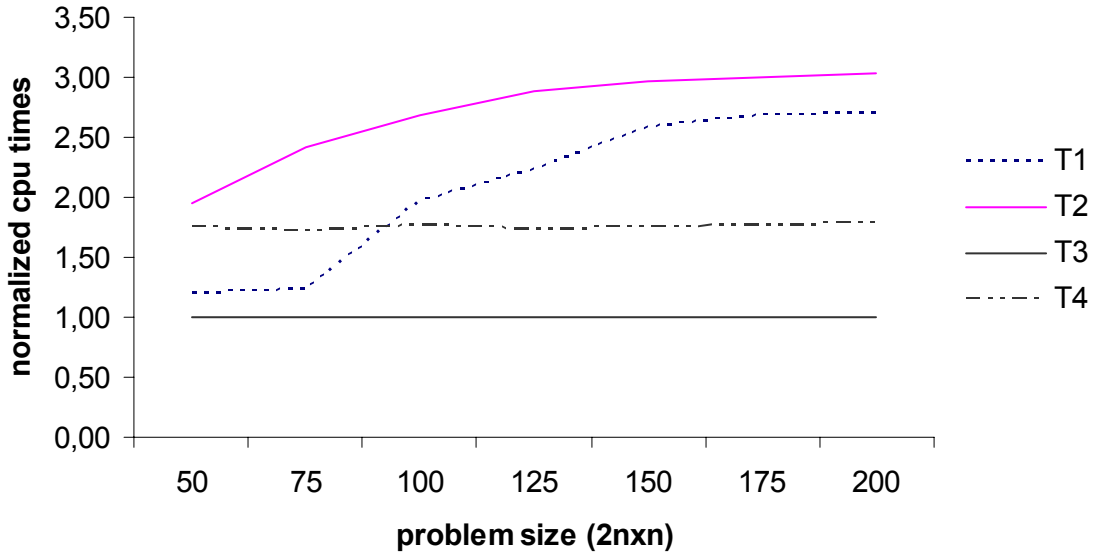
2nxn	T1		T2		T3		T4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	250,2	9,94	243,3	16,27	182,4	8,33	268,8	14,52
75	497,8	31,48	448,2	62,03	297,9	25,65	428,4	43,85
100	796,9	106,19	622,6	145,41	403,7	54,21	579,3	95,49
125	1.138,1	230,14	840,5	295,94	527,8	102,81	741,2	178,18
150	1.561,6	443,12	1.020,8	510,30	643,2	171,62	901,1	301,06
175	2.045,2	698,63	1.218,3	781,29	761,8	260,31	1.061,4	459,66
200	2.701,4	995,08	1.478,2	1.115,48	877,3	367,70	1.323,1	655,04

Πίνακας 6.27 - Σύγκριση των αλγορίθμων για την τρίτη κλάση δεδομένων, 2nxn

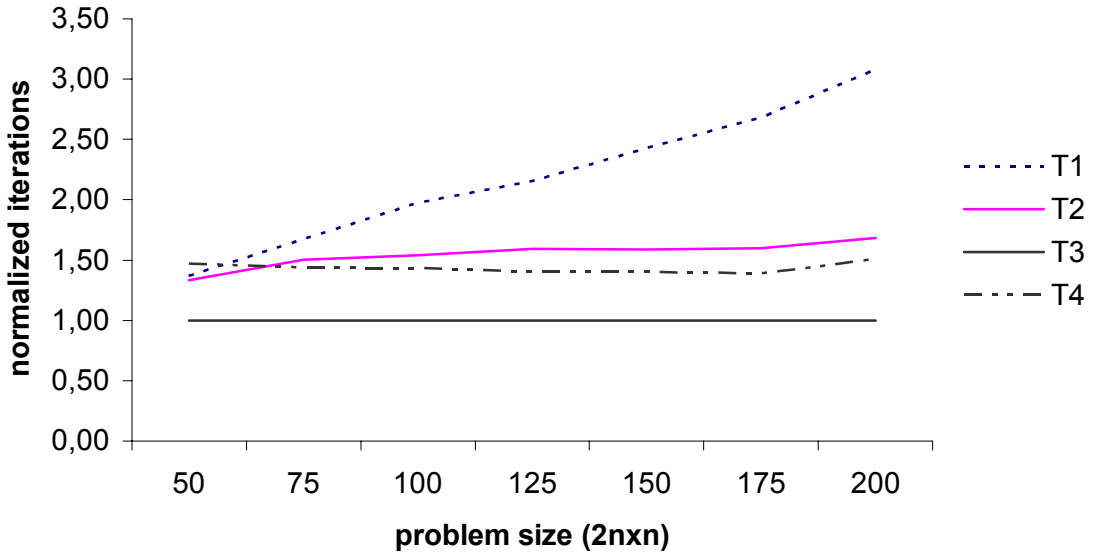
	T1/T2		T1/T3		T1/T4		T2/T3		T2/T4		T4/T3	
2nxn	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	1,028	0,611	1,372	1,193	0,931	0,685	1,334	1,953	0,905	0,685	1,474	1,743
75	1,111	0,507	1,671	1,227	1,162	0,718	1,505	2,418	1,046	0,718	1,438	1,710
100	1,280	0,730	1,974	1,959	1,376	1,112	1,542	2,682	1,075	1,112	1,435	1,761
125	1,354	0,778	2,156	2,238	1,536	1,292	1,592	2,879	1,134	1,292	1,404	1,733
150	1,530	0,868	2,428	2,582	1,733	1,472	1,587	2,973	1,133	1,472	1,401	1,754
175	1,679	0,894	2,685	2,684	1,927	1,520	1,599	3,001	1,148	1,520	1,393	1,766
200	1,827	0,892	3,079	2,706	2,042	1,519	1,685	3,034	1,117	1,519	1,508	1,781

Πίνακας 6.28 - Κανονικοποιημένα αποτελέσματα για την τρίτη κλάση δεδομένων, 2nxn

2nxn	T1		T2		T3		T4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	1,37	1,19	1,33	1,95	1,00	1,00	1,47	1,74
75	1,67	1,23	1,50	2,42	1,00	1,00	1,44	1,71
100	1,97	1,96	1,54	2,68	1,00	1,00	1,43	1,76
125	2,16	2,24	1,59	2,88	1,00	1,00	1,40	1,73
150	2,43	2,58	1,59	2,97	1,00	1,00	1,40	1,75
175	2,68	2,68	1,60	3,00	1,00	1,00	1,39	1,77
200	3,08	2,71	1,68	3,03	1,00	1,00	1,51	1,78



Εικόνα 6.16 – Κλάση 3:Το διάγραμμα των κανονικοποιημένων χρόνων, 2nxn



Εικόνα 6.17 – Κλάση 3:Το διάγραμμα των κανονικοποιημένων επαναλήψεων, 2nxn

Παρατηρούμε ότι στην εικόνα 6.15 έχουμε ανατροπή της μέχρι τώρα διάταξης. Αρχικά, ο αλγόριθμος T1, που μέχρι τώρα ήταν ο πιο αργός, έρχεται δεύτερος για μεγέθη μικρότερα του 90. Στη συνέχεια έρχεται τρίτος, παραχωρώντας τη θέση του στον T4. Ο αλγόριθμος T2 είναι μόνιμος τελευταίος. Συνεπώς, πρόκειται για "hard" προβλήματα.

6.4.2 Αποτελέσματα στο πρόβλημα Αντιστοίχησης

Τα μεγέθη προβλημάτων που χρησιμοποιήθηκαν είναι από 50 μέχρι 300 με βήμα 25 (συνολικά 11 μεγέθη). Τρέξαμε συνεπώς $7 \times 11 \times 10 = 770$ προβλήματα Αντιστοίχησης. Τα αποτελέσματα που πήραμε για κάθε κλάση ξεχωριστά -και τα αντίστοιχα διαγράμματα- φαίνονται στα παρακάτω σχήματα. Διαγράμματα θα παρουσιασθούν μόνο για τα κανονικοποιημένα αποτελέσματα

ΚΛΑΣΗ 1

Πίνακας 6.29 – Υπολογιστικά Αποτελέσματα της πρώτης κλάσης δεδομένων[5]

n	A1		A2		A3		A4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	272,0	10,68	149,8	6,07	67,3	3,78	116,2	3,97
75	497,1	30,30	295,1	23,98	112,2	10,57	193,3	12,79
100	769,6	64,40	430,4	57,66	159,5	21,86	261,9	27,92
125	1.045,9	113,33	589,2	118,39	206,9	38,89	335,6	53,03
150	1.381,8	184,07	786,9	218,65	250,0	61,33	420,3	90,63
175	1.717,0	274,41	998,6	368,58	295,7	91,08	493,9	142,88
200	2.086,3	385,51	1.238,4	573,10	352,1	130,83	561,2	206,18
225	2.454,7	525,75	1.481,9	869,59	398,9	179,71	660,0	301,69
250	2.878,7	694,95	1.604,6	1.138,26	460,6	240,82	728,4	405,67
275	3.272,5	905,18	1.883,4	1.593,06	505,6	307,29	807,7	536,61
300	3.638,7	1.124,43	2.195,7	2.186,33	562,6	368,11	894,6	696,81

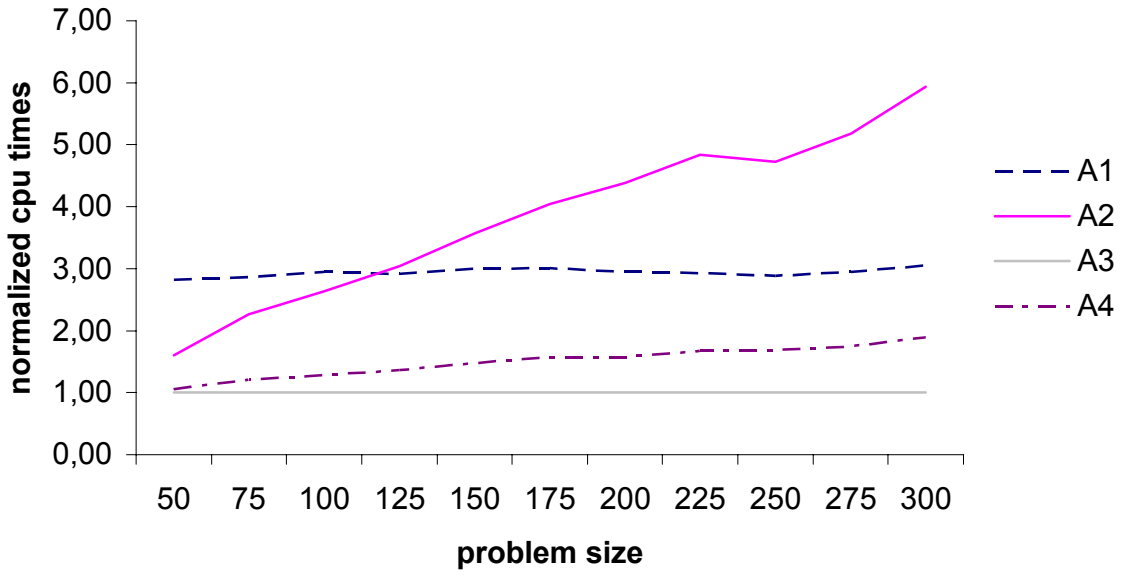
Πίνακας 6.30 – Σύγκριση των αλγορίθμων για την πρώτη κλάση δεδομένων

	A1/A2		A1/A3		A1/A4		A2/A3		A2/A4		A4/A3	
n	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	1,816	1,759	4,042	2,825	2,341	2,690	2,226	1,606	1,289	1,529	1,727	1,050
75	1,685	1,264	4,430	2,867	2,572	2,369	2,630	2,269	1,527	1,875	1,723	1,210
100	1,788	1,117	4,825	2,946	2,939	2,307	2,698	2,638	1,643	2,065	1,642	1,277
125	1,775	0,957	5,055	2,914	3,117	2,137	2,848	3,044	1,756	2,233	1,622	1,364
150	1,756	0,842	5,527	3,001	3,288	2,031	3,148	3,565	1,872	2,413	1,681	1,478
175	1,719	0,745	5,807	3,013	3,476	1,921	3,377	4,047	2,022	2,580	1,670	1,569
200	1,685	0,673	5,925	2,947	3,718	1,870	3,517	4,380	2,207	2,780	1,594	1,576
225	1,656	0,605	6,154	2,926	3,719	1,743	3,715	4,839	2,245	2,882	1,655	1,679
250	1,794	0,611	6,250	2,886	3,952	1,713	3,484	4,727	2,203	2,806	1,581	1,685
275	1,738	0,568	6,473	2,946	4,052	1,687	3,725	5,184	2,332	2,969	1,598	1,746
300	1,657	0,514	6,468	3,055	4,067	1,614	3,903	5,939	2,454	3,138	1,590	1,893

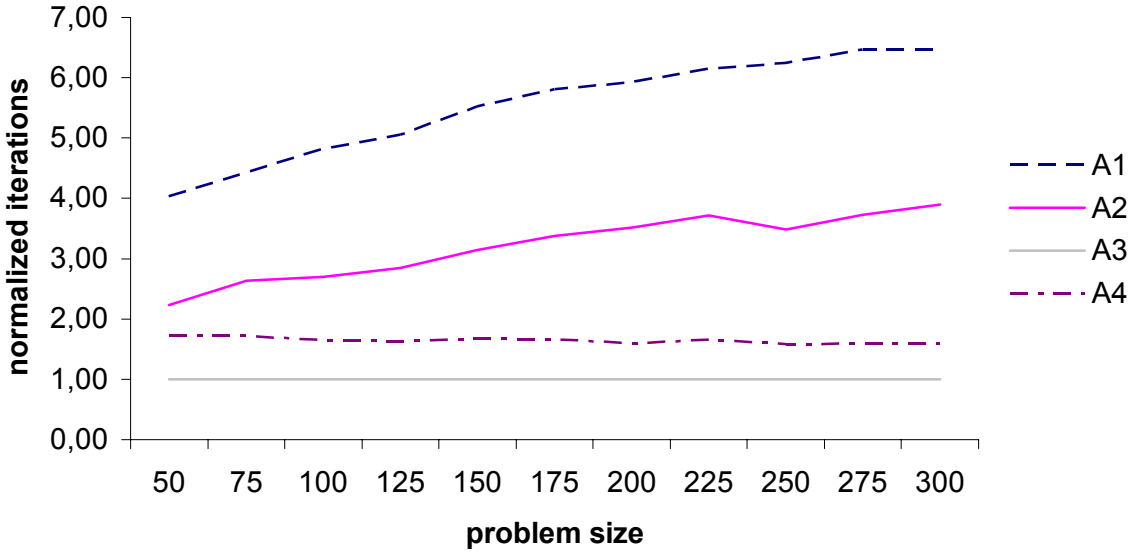
Πίνακας 6.31 – Κανονικοποιημένα αποτελέσματα για την πρώτη κλάση

n	A1		A2		A3		A4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	4,04	2,83	2,23	1,61	1,00	1,00	1,73	1,05
75	4,43	2,87	2,63	2,27	1,00	1,00	1,72	1,21
100	4,83	2,95	2,70	2,64	1,00	1,00	1,64	1,28
125	5,06	2,91	2,85	3,04	1,00	1,00	1,62	1,36
150	5,53	3,00	3,15	3,57	1,00	1,00	1,68	1,48
175	5,81	3,01	3,38	4,05	1,00	1,00	1,67	1,57
200	5,93	2,95	3,52	4,38	1,00	1,00	1,59	1,58
225	6,15	2,93	3,71	4,84	1,00	1,00	1,65	1,68
250	6,25	2,89	3,48	4,73	1,00	1,00	1,58	1,68
275	6,47	2,95	3,73	5,18	1,00	1,00	1,60	1,75
300	6,47	3,05	3,90	5,94	1,00	1,00	1,59	1,89

[5] Οι χρόνοι εκφράζονται σε δευτερόλεπτα (sec)



Εικόνα 6.18 – Κλάση 1: Το διάγραμμα των κανονικοποιημένων χρόνων



Εικόνα 6.19– Κλάση 1: Το διάγραμμα των κανονικοποιημένων επαναλήψεων

Παρατηρούμε ότι ο αλγόριθμος A3 είναι ανώτερος και σε χρόνο CPU άλλα και σε αριθμό επαναλήψεων. Συγκεκριμένα, είναι τρεις φορές καλύτερος σε χρόνο CPU από τον αλγόριθμο A1 ενώ για μεγάλα μεγέθη η διαφορά στον αριθμό επαναλήψεων είναι εξαπλάσια. Επίσης, ενώ αρχικά ο αλγόριθμος A1(simplex) φαίνεται να είναι χειρότερος από άποψη χρόνου σε σχέση με όλους τους άλλους αλγορίθμους εξωτερικών σημείων(τέταρτος στην κατάταξή), για $n \geq 110$ έρχεται τρίτος στην κατάταξη, παραχωρώντας τη θέση του στον αλγόριθμο A2. Ο αλγόριθμος A4 παραμένει σταθερά δεύτερος.

ΚΛΑΣΗ 2

Πίνακας 6.32 - Υπολογιστικά Αποτελέσματα της δεύτερης κλάσης δεδομένων

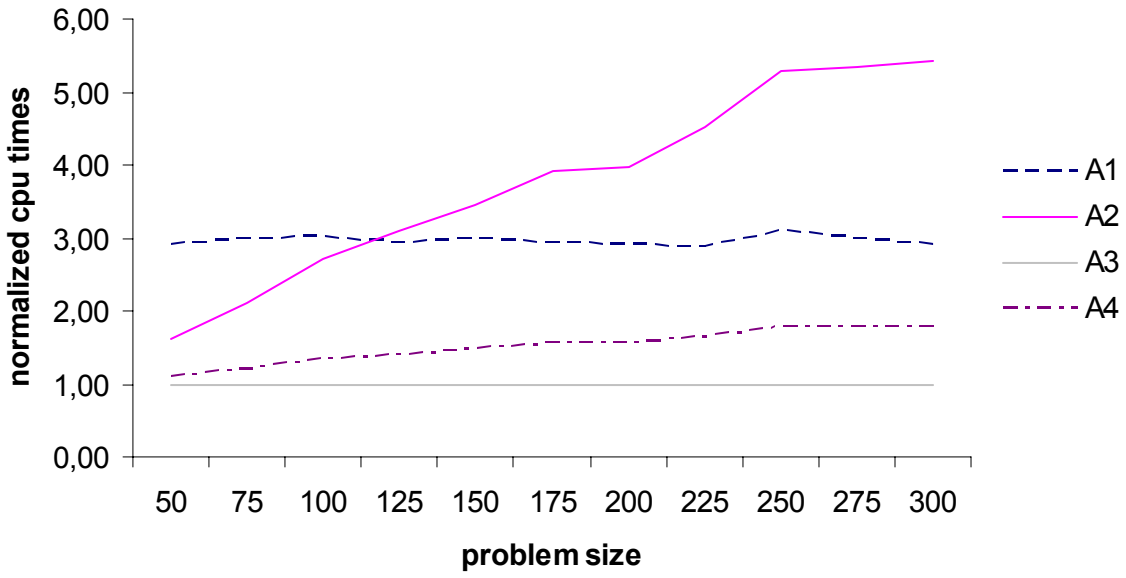
n	A1		A2		A3		A4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	277,9	10,92	149,8	6,07	67,0	3,77	120,9	4,12
75	508,0	30,99	272,5	21,85	110,4	10,42	188,1	12,58
100	773,2	65,04	438,9	58,19	156,3	21,55	266,5	28,68
125	1.059,4	114,69	606,2	120,96	209,7	39,18	343,8	54,95
150	1.399,8	187,83	777,6	216,67	253,8	62,69	416,7	92,03
175	1.680,8	269,57	979,9	361,29	298,3	91,97	489,2	142,49
200	2.087,8	390,51	1.141,5	535,76	358,2	134,56	574,0	211,91
225	2.461,2	532,21	1.410,9	831,36	406,9	184,17	662,6	304,99
250	2.833,9	691,48	1.664,8	1.186,93	424,4	224,20	710,2	400,31
275	3.187,4	884,87	1.874,4	1.588,66	477,4	297,13	793,2	526,70
300	3.643,6	1.144,10	2.101,5	2.126,59	545,2	392,82	883,6	699,81

Πίνακας 6.33 – Σύγκριση των αλγορίθμων για την δεύτερη κλάση δεδομένων

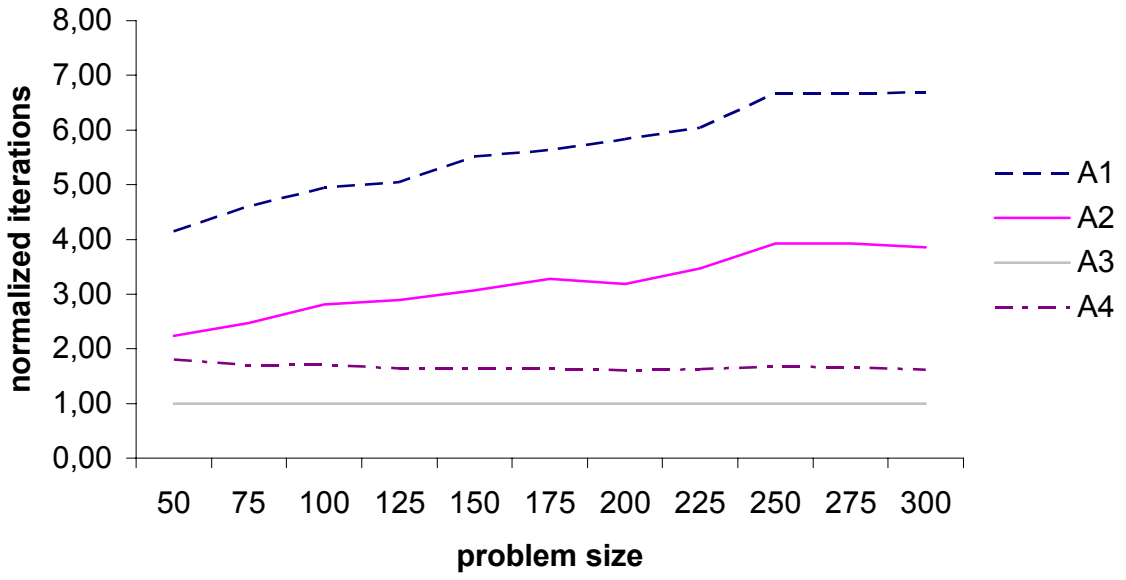
	A1/A2		A1/A3		A1/A4		A2/A3		A2/A4		A4/A3	
n	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	1,855	1,799	4,148	2,897	2,299	2,650	2,236	1,610	1,239	1,473	1,804	1,093
75	1,864	1,418	4,601	2,974	2,701	2,463	2,468	2,097	1,449	1,737	1,704	1,207
100	1,762	1,118	4,947	3,018	2,901	2,268	2,808	2,700	1,647	2,029	1,705	1,331
125	1,748	0,948	5,052	2,927	3,081	2,087	2,891	3,087	1,763	2,201	1,639	1,403
150	1,800	0,867	5,515	2,996	3,359	2,041	3,064	3,456	1,866	2,354	1,642	1,468
175	1,715	0,746	5,635	2,931	3,436	1,892	3,285	3,928	2,003	2,536	1,640	1,549
200	1,829	0,729	5,829	2,902	3,637	1,843	3,187	3,982	1,989	2,528	1,602	1,575
225	1,744	0,640	6,049	2,890	3,714	1,745	3,467	4,514	2,129	2,726	1,628	1,656
250	1,702	0,583	6,677	3,084	3,990	1,727	3,923	5,294	2,344	2,965	1,673	1,786
275	1,700	0,557	6,677	2,978	4,018	1,680	3,926	5,347	2,363	3,016	1,661	1,773
300	1,734	0,538	6,683	2,913	4,124	1,635	3,855	5,414	2,378	3,039	1,621	1,782

Πίνακας 6.34 – Κανονικοποιημενα αποτελέσματα για τη δεύτερη κλάση δεδομένων

n	A1		A2		A3		A4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	4,15	2,90	2,24	1,61	1,00	1,00	1,80	1,09
75	4,60	2,97	2,47	2,10	1,00	1,00	1,70	1,21
100	4,95	3,02	2,81	2,70	1,00	1,00	1,71	1,33
125	5,05	2,93	2,89	3,09	1,00	1,00	1,64	1,40
150	5,52	3,00	3,06	3,46	1,00	1,00	1,64	1,47
175	5,63	2,93	3,28	3,93	1,00	1,00	1,64	1,55
200	5,83	2,90	3,19	3,98	1,00	1,00	1,60	1,57
225	6,05	2,89	3,47	4,51	1,00	1,00	1,63	1,66
250	6,68	3,08	3,92	5,29	1,00	1,00	1,67	1,79
275	6,68	2,98	3,93	5,35	1,00	1,00	1,66	1,77
300	6,68	2,91	3,85	5,41	1,00	1,00	1,62	1,78



Εικόνα 6.20 - Κλάση 2: Το διάγραμμα των κανονικοποιημένων χρόνων



Εικόνα 6.21 – Κλάση 2: Το διάγραμμα των κανονικοποιημένων επαναλήψεων

Και για αυτήν την κλάση παρατηρούμε ότι επιβεβαιώνονται τα αποτελέσματα που πήραμε προηγουμένως. Ο αλγόριθμος A3 παραμένει σταθερά πρώτος.

ΚΛΑΣΗ 3

Πίνακας 6.35 - Υπολογιστικά Αποτελέσματα της τρίτης κλάσης δεδομένων

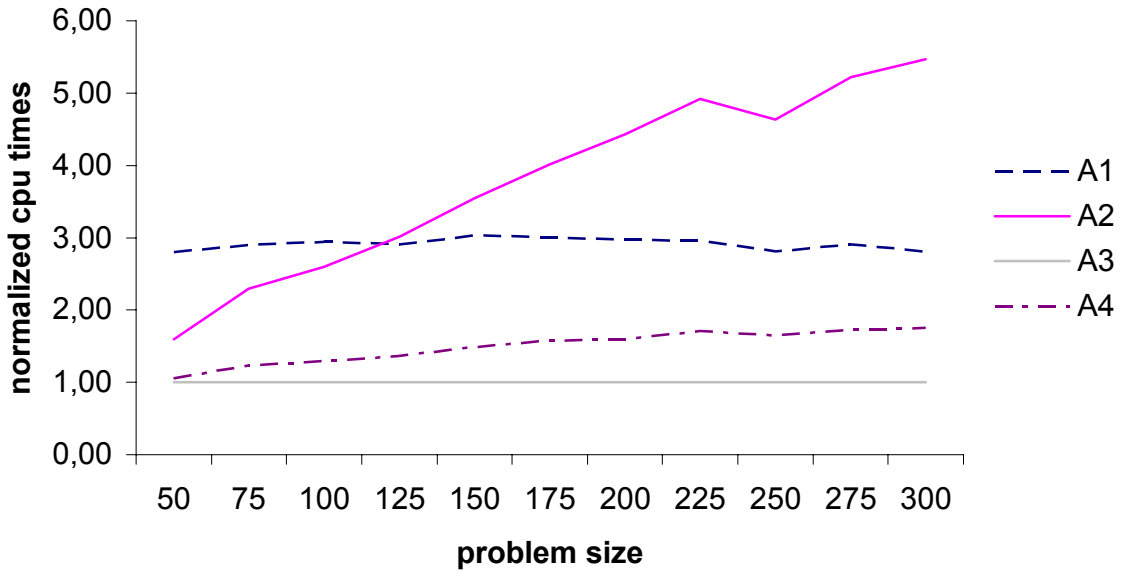
n	A1		A2		A3		A4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	271,2	10,65	149,8	6,06	67,6	3,80	116,7	4,02
75	494,3	30,25	296,3	23,96	112,5	10,44	193,4	12,88
100	772,4	64,48	430,0	56,88	159,4	21,94	261,7	28,38
125	1.053,5	113,01	589,8	117,18	207,5	38,87	334,9	53,03
150	1.393,9	183,98	787,5	215,47	249,7	60,71	419,8	90,27
175	1.720,3	272,36	997,8	364,70	295,7	90,74	494,1	143,10
200	2.086,3	386,52	1.238,7	576,93	352,3	130,17	560,6	207,40
225	2.469,8	524,83	1.484,5	870,56	398,4	177,13	660,0	302,38
250	2.867,0	692,57	1.604,1	1.143,06	461,3	246,51	728,6	407,00
275	3.269,3	906,20	1.885,8	1.625,69	505,7	311,36	806,5	538,51
300	3.636,8	1.128,25	2.194,4	2.197,83	563,3	401,86	894,4	704,26

Πίνακας 6.36 - Σύγκριση των αλγορίθμων για την τρίτη κλάση δεδομένων

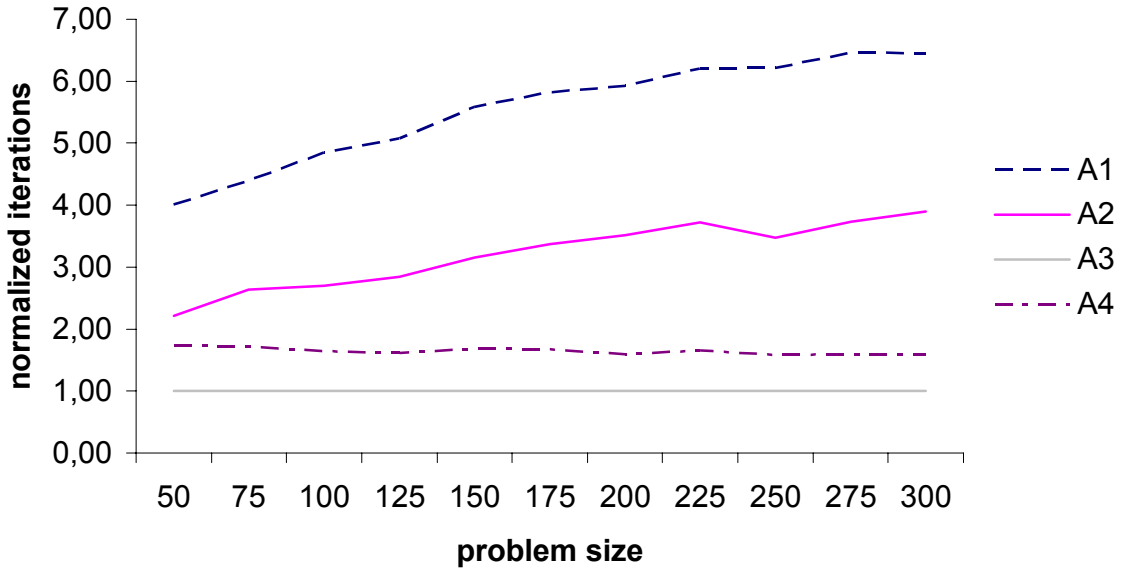
	A1/A2		A1/A3		A1/A4		A2/A3		A2/A4		A4/A3	
n	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	1,810	1,757	4,012	2,803	2,324	2,649	2,216	1,595	1,284	1,507	1,726	1,058
75	1,668	1,263	4,394	2,898	2,556	2,349	2,634	2,295	1,532	1,860	1,719	1,234
100	1,796	1,134	4,846	2,939	2,951	2,272	2,698	2,593	1,643	2,004	1,642	1,294
125	1,786	0,964	5,077	2,907	3,146	2,131	2,842	3,015	1,761	2,210	1,614	1,364
150	1,770	0,854	5,582	3,030	3,320	2,038	3,154	3,549	1,876	2,387	1,681	1,487
175	1,724	0,747	5,818	3,002	3,482	1,903	3,374	4,019	2,019	2,549	1,671	1,577
200	1,684	0,670	5,922	2,969	3,722	1,864	3,516	4,432	2,210	2,782	1,591	1,593
225	1,664	0,603	6,199	2,963	3,742	1,736	3,726	4,915	2,249	2,879	1,657	1,707
250	1,787	0,606	6,215	2,810	3,935	1,702	3,477	4,637	2,202	2,809	1,579	1,651
275	1,734	0,557	6,465	2,910	4,054	1,683	3,729	5,221	2,338	3,019	1,595	1,730
300	1,657	0,513	6,456	2,808	4,066	1,602	3,896	5,469	2,453	3,121	1,588	1,753

Πίνακας 6.37 - Κανονικοποιημενα αποτελέσματα για τη τρίτη κλάση δεδομένων

n	A1		A2		A3		A4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	4,01	2,80	2,22	1,59	1,00	1,00	1,73	1,06
75	4,39	2,90	2,63	2,30	1,00	1,00	1,72	1,23
100	4,85	2,94	2,70	2,59	1,00	1,00	1,64	1,29
125	5,08	2,91	2,84	3,01	1,00	1,00	1,61	1,36
150	5,58	3,03	3,15	3,55	1,00	1,00	1,68	1,49
175	5,82	3,00	3,37	4,02	1,00	1,00	1,67	1,58
200	5,92	2,97	3,52	4,43	1,00	1,00	1,59	1,59
225	6,20	2,96	3,73	4,91	1,00	1,00	1,66	1,71
250	6,22	2,81	3,48	4,64	1,00	1,00	1,58	1,65
275	6,46	2,91	3,73	5,22	1,00	1,00	1,59	1,73
300	6,46	2,81	3,90	5,47	1,00	1,00	1,59	1,75



Εικόνα 6.22 - Κλάση 3: Το διάγραμμα των κανονικοποιημένων χρόνων



Εικόνα 6.23 – Κλάση 3: Το διάγραμμα των κανονικοποιημένων επαναλήψεων

ΚΛΑΣΗ 4

Πίνακας 6.38 - Υπολογιστικά Αποτελέσματα της τέταρτης κλάσης δεδομένων

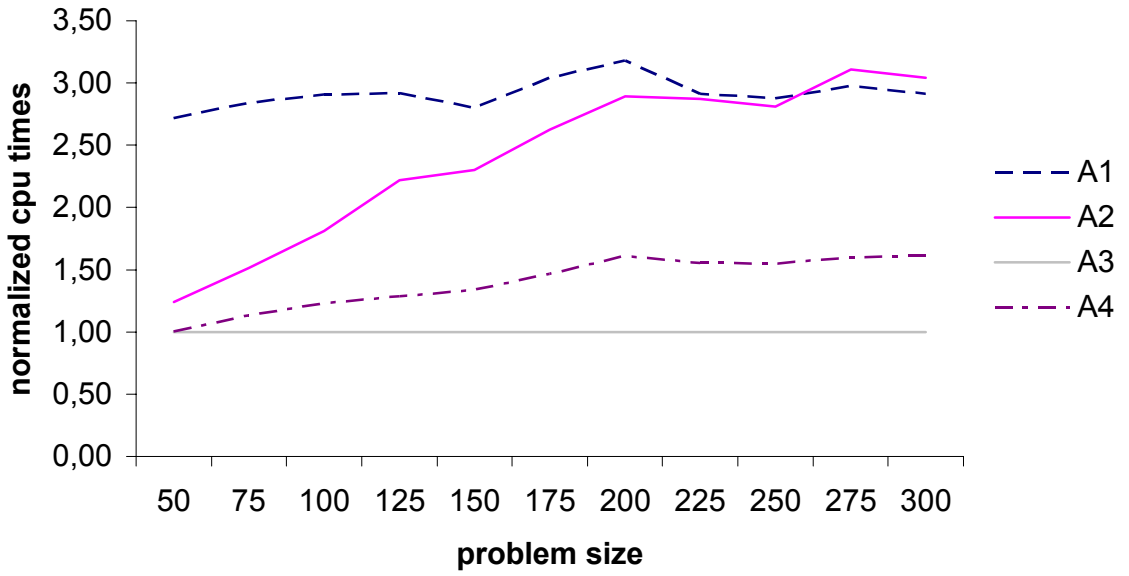
n	A1		A2		A3		A4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	276,8	11,31	121,0	5,15	69,7	4,16	114,7	4,18
75	498,5	31,79	200,3	16,94	117,2	11,19	180,8	12,67
100	767,8	64,59	294,0	40,26	158,0	22,23	246,3	27,37
125	1.043,1	113,52	417,5	86,29	202,5	38,91	304,3	49,97
150	1.333,9	179,27	511,3	147,25	255,7	64,06	377,4	85,62
175	1.727,4	275,04	629,2	237,54	290,0	90,44	449,9	132,75
200	2.084,9	393,74	727,4	358,43	351,0	123,87	518,0	199,76
225	2.415,8	520,43	838,4	513,41	392,0	178,80	598,0	277,86
250	2.840,7	697,85	929,0	681,19	453,2	242,58	663,6	375,87
275	3.215,6	902,49	1.066,2	942,99	481,3	303,41	709,5	485,54
300	3.666,7	1.162,89	1.144,2	1.214,31	546,5	399,51	793,0	642,89

Πίνακας 6.39 - Σύγκριση των αλγορίθμων για την τέταρτη κλάση δεδομένων

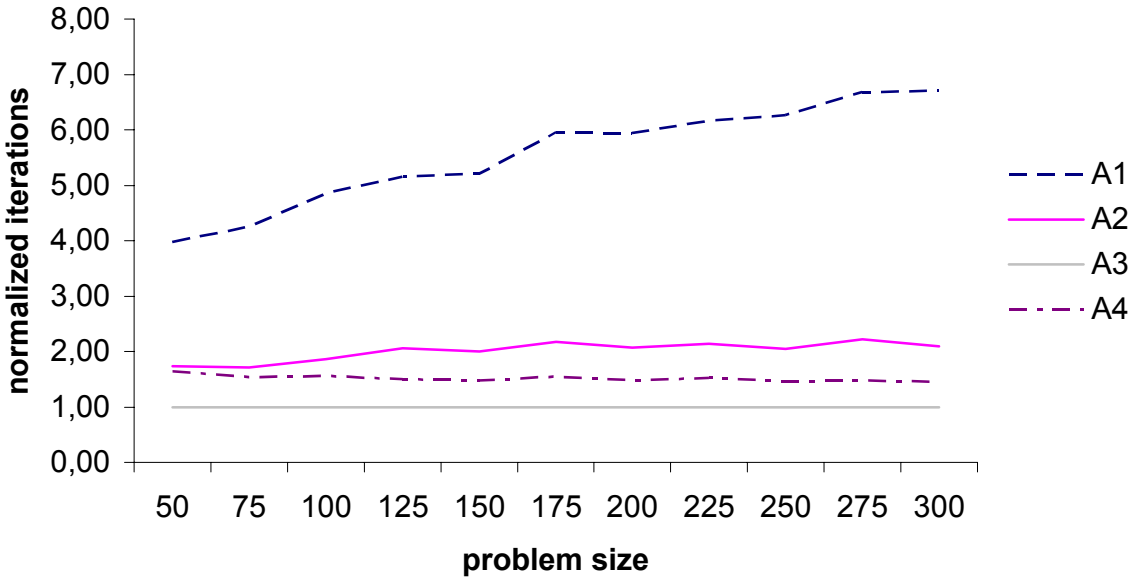
	A1/A2		A1/A3		A1/A4		A2/A3		A2/A4		A4/A3	
n	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	2,288	2,196	3,971	2,719	2,413	2,706	1,736	1,238	1,055	1,232	1,646	1,005
75	2,489	1,877	4,253	2,841	2,757	2,509	1,709	1,514	1,108	1,337	1,543	1,132
100	2,612	1,604	4,859	2,906	3,117	2,360	1,861	1,811	1,194	1,471	1,559	1,231
125	2,498	1,316	5,151	2,918	3,428	2,272	2,062	2,218	1,372	1,727	1,503	1,284
150	2,609	1,217	5,217	2,798	3,534	2,094	2,000	2,299	1,355	1,720	1,476	1,337
175	2,745	1,158	5,957	3,041	3,840	2,072	2,170	2,626	1,399	1,789	1,551	1,468
200	2,866	1,099	5,940	3,179	4,025	1,971	2,072	2,894	1,404	1,794	1,476	1,613
225	2,881	1,014	6,163	2,911	4,040	1,873	2,139	2,871	1,402	1,848	1,526	1,554
250	3,058	1,024	6,268	2,877	4,281	1,857	2,050	2,808	1,400	1,812	1,464	1,549
275	3,016	0,957	6,681	2,974	4,532	1,859	2,215	3,108	1,503	1,942	1,474	1,600
300	3,205	0,958	6,709	2,911	4,624	1,809	2,094	3,039	1,443	1,889	1,451	1,609

Πίνακας 6.40 - Κανονικοποιημενα αποτελέσματα για τη τέταρτη κλάση δεδομένων

n	A1		A2		A3		A4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	3,97	2,72	1,74	1,24	1,00	1,00	1,65	1,00
75	4,25	2,84	1,71	1,51	1,00	1,00	1,54	1,13
100	4,86	2,91	1,86	1,81	1,00	1,00	1,56	1,23
125	5,15	2,92	2,06	2,22	1,00	1,00	1,50	1,28
150	5,22	2,80	2,00	2,30	1,00	1,00	1,48	1,34
175	5,96	3,04	2,17	2,63	1,00	1,00	1,55	1,47
200	5,94	3,18	2,07	2,89	1,00	1,00	1,48	1,61
225	6,16	2,91	2,14	2,87	1,00	1,00	1,53	1,55
250	6,27	2,88	2,05	2,81	1,00	1,00	1,46	1,55
275	6,68	2,97	2,22	3,11	1,00	1,00	1,47	1,60
300	6,71	2,91	2,09	3,04	1,00	1,00	1,45	1,61



Εικόνα 6.24 - Κλάση 4: Το διάγραμμα των κανονικοποιημένων χρόνων



Εικόνα 6.25 - Κλάση 4: Το διάγραμμα των κανονικοποιημένων επαναλήψεων

Παρατηρούμε ότι για αυτή την κλάση δεδομένων υπάρχει μια ανατροπή της μέχρι τώρα κατάστασης. Ο αλγόριθμος Α2 εμφανίζεται πολύ καλύτερος και ανταγωνίζεται των αλγόριθμο Α1 ακόμη και σε μεγάλα μεγέθη προβλημάτων. Γενικά, αυτή η κλάση μπορούμε να πούμε ότι ευνοεί τους αλγορίθμους εξωτερικών σημείων.

ΚΛΑΣΗ 5

Πίνακας 6.41 - Υπολογιστικά Αποτελέσματα της πέμπτης κλάσης δεδομένων

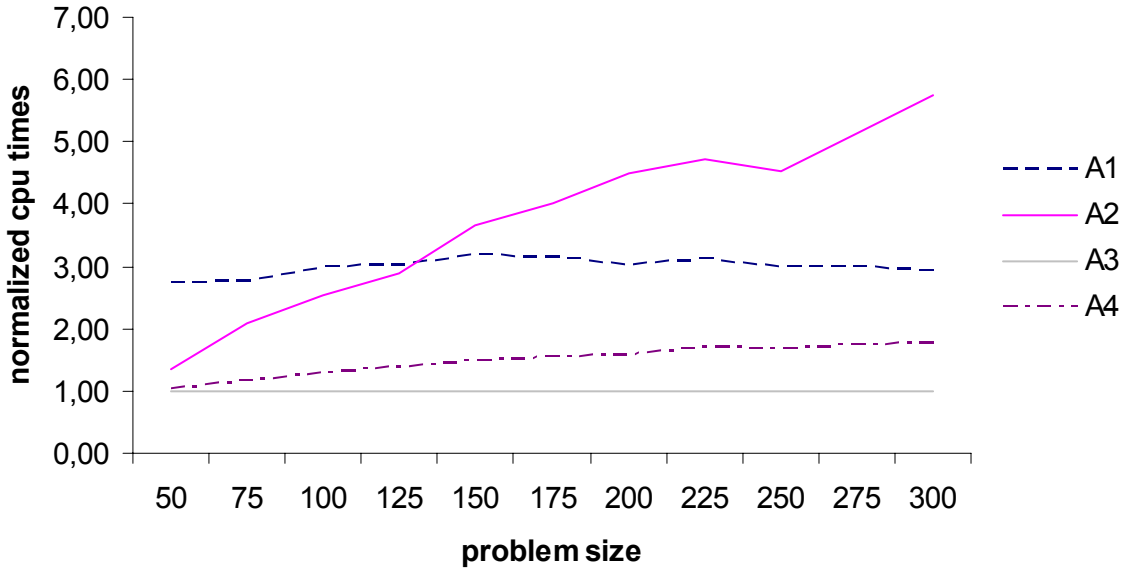
n	A1		A2		A3		A4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	291,4	11,54	137,8	5,63	75,3	4,22	129,0	4,38
75	511,8	31,18	293,4	23,46	124,2	11,32	200,1	12,94
100	789,8	66,24	423,0	56,09	159,7	22,14	265,9	28,33
125	1.088,2	116,90	568,6	111,86	207,3	38,67	340,5	53,24
150	1.450,1	193,01	797,6	221,96	249,5	60,81	412,9	90,72
175	1.777,2	282,36	987,8	360,99	294,3	90,05	490,3	140,14
200	2.146,4	403,02	1.266,7	599,44	356,3	133,27	569,1	208,76
225	2.533,6	540,40	1.420,5	820,24	391,5	174,22	644,4	293,88
250	2.910,5	712,15	1.524,7	1.081,55	458,3	238,65	724,8	397,42
275	3.351,9	927,34	1.902,3	1.601,05	504,4	311,13	811,0	538,34
300	3.709,9	1.159,88	2.258,7	2.295,93	559,5	398,41	902,0	708,37

Πίνακας 6.42 - Σύγκριση των αλγορίθμων για την πέμπτη κλάση δεδομένων

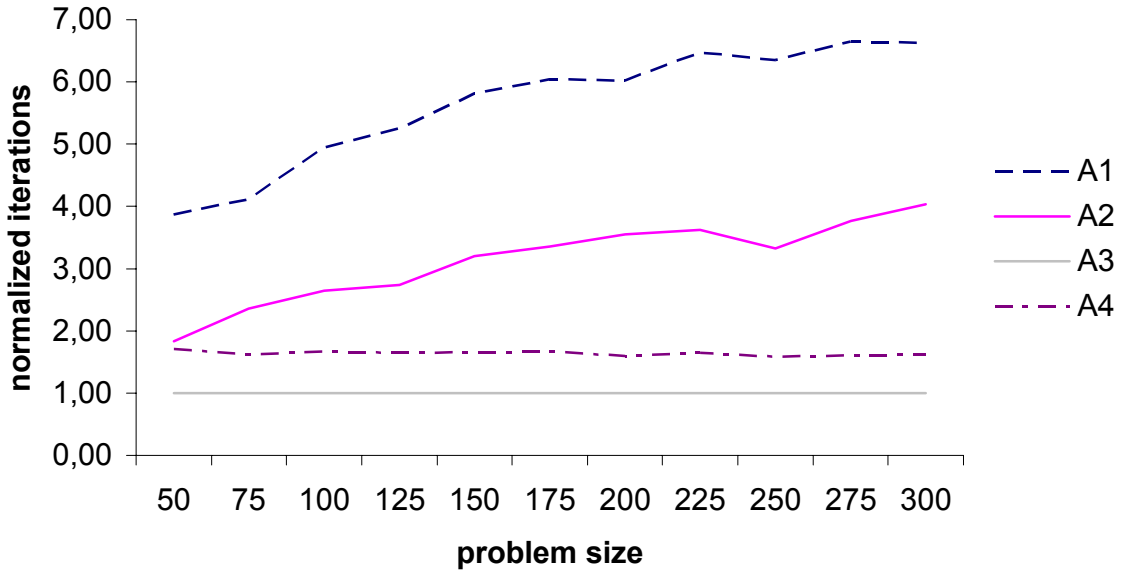
	A1/A2		A1/A3		A1/A4		A2/A3		A2/A4		A4/A3	
n	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	2,115	2,050	3,870	2,735	2,259	2,635	1,830	1,334	1,068	1,285	1,713	1,038
75	1,744	1,329	4,121	2,754	2,558	2,410	2,362	2,072	1,466	1,813	1,611	1,143
100	1,867	1,181	4,946	2,992	2,970	2,338	2,649	2,533	1,591	1,980	1,665	1,280
125	1,914	1,045	5,249	3,023	3,196	2,196	2,743	2,893	1,670	2,101	1,643	1,377
150	1,818	0,870	5,812	3,174	3,512	2,128	3,197	3,650	1,932	2,447	1,655	1,492
175	1,799	0,782	6,039	3,136	3,625	2,015	3,356	4,009	2,015	2,576	1,666	1,556
200	1,694	0,672	6,024	3,024	3,772	1,931	3,555	4,498	2,226	2,871	1,597	1,566
225	1,784	0,659	6,472	3,102	3,932	1,839	3,628	4,708	2,204	2,791	1,646	1,687
250	1,909	0,658	6,351	2,984	4,016	1,792	3,327	4,532	2,104	2,721	1,581	1,665
275	1,762	0,579	6,645	2,981	4,133	1,723	3,771	5,146	2,346	2,974	1,608	1,730
300	1,642	0,505	6,631	2,911	4,113	1,637	4,037	5,763	2,504	3,241	1,612	1,778

Πίνακας 6.43 - Κανονικοποιημενα αποτελέσματα για τη πέμπτη κλάση δεδομένων

n	A1		A2		A3		A4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	3,87	2,73	1,83	1,33	1,00	1,00	1,71	1,04
75	4,12	2,75	2,36	2,07	1,00	1,00	1,61	1,14
100	4,95	2,99	2,65	2,53	1,00	1,00	1,66	1,28
125	5,25	3,02	2,74	2,89	1,00	1,00	1,64	1,38
150	5,81	3,17	3,20	3,65	1,00	1,00	1,65	1,49
175	6,04	3,14	3,36	4,01	1,00	1,00	1,67	1,56
200	6,02	3,02	3,56	4,50	1,00	1,00	1,60	1,57
225	6,47	3,10	3,63	4,71	1,00	1,00	1,65	1,69
250	6,35	2,98	3,33	4,53	1,00	1,00	1,58	1,67
275	6,65	2,98	3,77	5,15	1,00	1,00	1,61	1,73
300	6,63	2,91	4,04	5,76	1,00	1,00	1,61	1,78



Εικόνα 6.26 - Κλάση 5: Το διάγραμμα των κανονικοποιημένων χρόνων



Εικόνα 6.27 - Κλάση 5: Το διάγραμμα των κανονικοποιημένων επαναλήψεων

Παρατηρούμε ότι σε αυτήν την κλάση επαναφέρεται η κατάσταση που των κλάσεων 1 – 4.

ΚΛΑΣΗ 6

Πίνακας 6.44 - Υπολογιστικά Αποτελέσματα της έκτης κλάσης δεδομένων

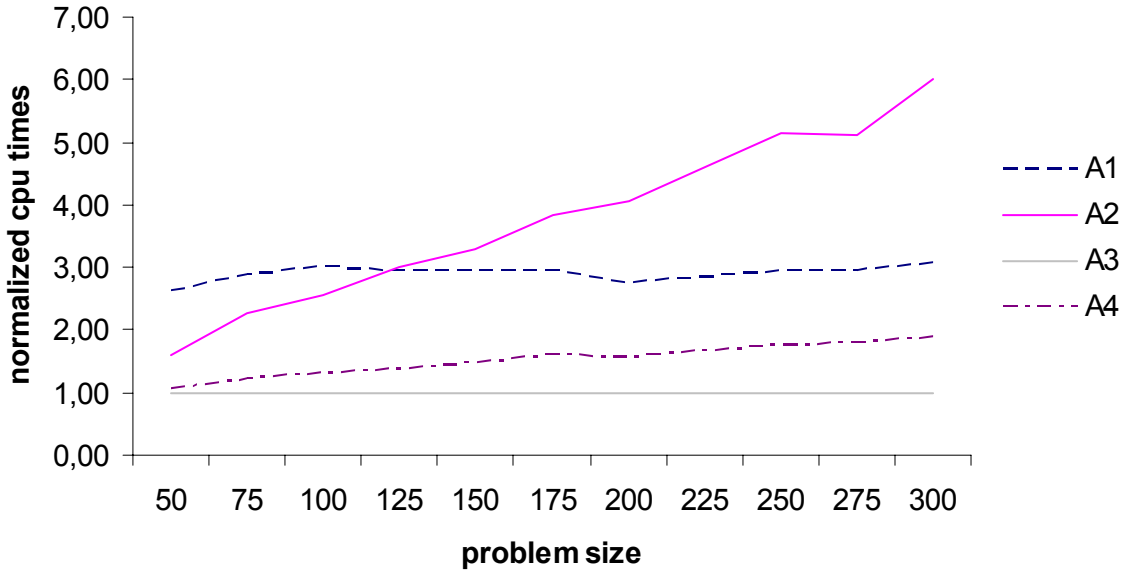
n	A1		A2		A3		A4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	271,9	10,56	158,8	6,42	72,0	4,01	122,2	4,19
75	496,8	29,89	292,2	23,59	111,9	10,42	192,0	12,52
100	778,1	64,67	417,9	54,89	156,1	21,46	264,0	28,09
125	1.060,7	113,39	589,0	115,90	207,0	38,75	335,4	53,01
150	1.381,7	182,23	749,0	203,31	254,6	61,86	416,6	90,54
175	1.713,5	269,46	954,0	348,62	308,1	91,14	500,3	144,72
200	2.035,4	377,06	1.180,5	558,19	368,3	137,03	580,6	214,88
225	2.426,2	515,65	1.412,4	828,29	402,2	180,26	652,3	298,33
250	2.815,3	681,92	1.674,2	1.196,02	460,0	231,81	736,9	410,99
275	3.234,9	895,04	1.819,0	1.556,78	511,0	304,91	810,9	542,33
300	3.692,6	1.138,80	2.191,5	2.232,49	558,2	372,18	895,3	702,63

Πίνακας 6.45 - Σύγκριση των αλγορίθμων για την έκτη κλάση δεδομένων

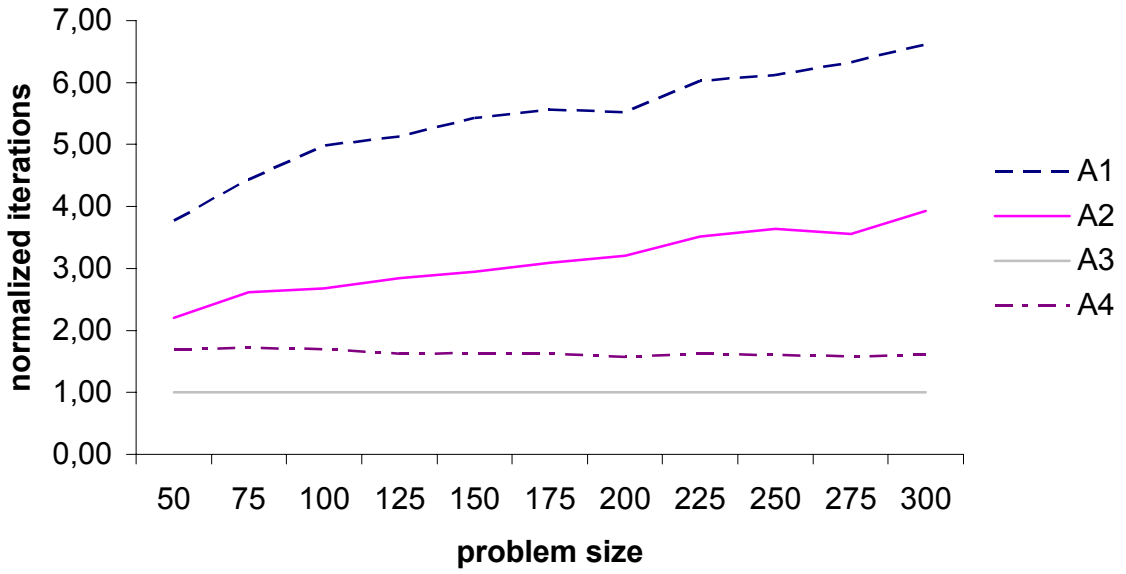
	A1/A2		A1/A3		A1/A4		A2/A3		A2/A4		A4/A3	
n	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	1,712	1,645	3,776	2,633	2,225	2,520	2,206	1,601	1,300	1,532	1,697	1,045
75	1,700	1,267	4,440	2,869	2,588	2,387	2,611	2,264	1,522	1,884	1,716	1,202
100	1,862	1,178	4,985	3,014	2,947	2,302	2,677	2,558	1,583	1,954	1,691	1,309
125	1,801	0,978	5,124	2,926	3,162	2,139	2,845	2,991	1,756	2,186	1,620	1,368
150	1,845	0,896	5,427	2,946	3,317	2,013	2,942	3,287	1,798	2,246	1,636	1,464
175	1,796	0,773	5,562	2,957	3,425	1,862	3,096	3,825	1,907	2,409	1,624	1,588
200	1,724	0,676	5,526	2,752	3,506	1,755	3,205	4,073	2,033	2,598	1,576	1,568
225	1,718	0,623	6,032	2,861	3,719	1,728	3,512	4,595	2,165	2,776	1,622	1,655
250	1,682	0,570	6,120	2,942	3,820	1,659	3,640	5,159	2,272	2,910	1,602	1,773
275	1,778	0,575	6,331	2,935	3,989	1,650	3,560	5,106	2,243	2,871	1,587	1,779
300	1,685	0,510	6,615	3,060	4,124	1,621	3,926	5,998	2,448	3,177	1,604	1,888

Πίνακας 6.46 - Κανονικοποιημενα αποτελέσματα για τη έκτη κλάση δεδομένων

n	A1		A2		A3		A4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	3,78	2,63	2,21	1,60	1,00	1,00	1,70	1,04
75	4,44	2,87	2,61	2,26	1,00	1,00	1,72	1,20
100	4,98	3,01	2,68	2,56	1,00	1,00	1,69	1,31
125	5,12	2,93	2,85	2,99	1,00	1,00	1,62	1,37
150	5,43	2,95	2,94	3,29	1,00	1,00	1,64	1,46
175	5,56	2,96	3,10	3,83	1,00	1,00	1,62	1,59
200	5,53	2,75	3,21	4,07	1,00	1,00	1,58	1,57
225	6,03	2,86	3,51	4,59	1,00	1,00	1,62	1,65
250	6,12	2,94	3,64	5,16	1,00	1,00	1,60	1,77
275	6,33	2,94	3,56	5,11	1,00	1,00	1,59	1,78
300	6,62	3,06	3,93	6,00	1,00	1,00	1,60	1,89



Εικόνα 6.28 - Κλάση 6: Το διάγραμμα των κανονικοποιημένων χρόνων



Εικόνα 6.29 - Κλάση 6: Το διάγραμμα των κανονικοποιημένων επαναλήψεων

ΚΛΑΣΗ 7

Πίνακας 6.47 - Υπολογιστικά Αποτελέσματα της έβδομης κλάσης δεδομένων

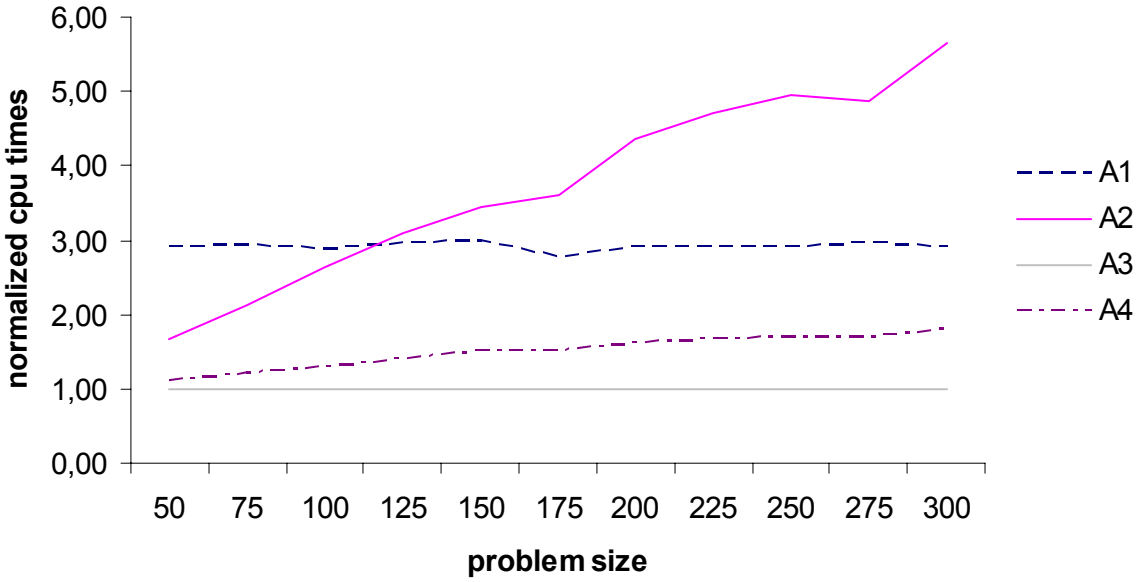
n	A1		A2		A3		A4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	276,2	10,67	151,9	6,14	67,0	3,68	118,5	4,03
75	503,2	30,34	272,6	22,04	111,0	10,35	186,8	12,41
100	776,6	64,20	438,5	58,38	165,2	22,25	271,4	29,00
125	1.064,4	113,51	595,8	119,06	206,0	38,41	342,1	53,86
150	1.385,9	181,89	754,7	209,94	251,5	61,07	421,6	92,08
175	1.681,2	264,90	944,8	346,35	312,8	96,00	503,6	144,82
200	2.054,1	379,82	1.195,9	566,50	348,8	130,14	566,0	208,80
225	2.429,7	519,62	1.433,7	839,91	394,3	178,51	646,3	296,67
250	2.855,4	695,14	1.665,2	1.176,15	450,2	238,13	725,8	402,53
275	3.226,9	929,79	1.816,5	1.537,23	501,3	314,81	799,1	532,83
300	3.630,9	1.121,47	2.144,5	2.187,64	552,7	386,51	887,9	698,78

Πίνακας 6.48 - Σύγκριση των αλγορίθμων για την έβδομη κλάση δεδομένων

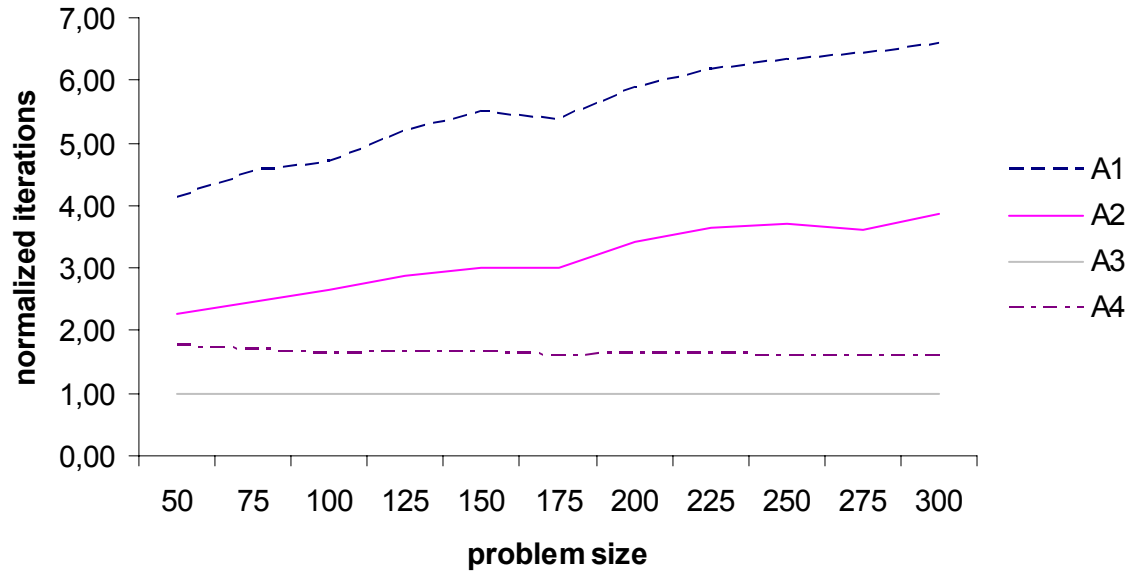
	A1/A2		A1/A3		A1/A4		A2/A3		A2/A4		A4/A3	
n	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	1,818	1,738	4,122	2,899	2,331	2,648	2,267	1,668	1,282	1,524	1,769	1,095
75	1,846	1,377	4,533	2,931	2,694	2,445	2,456	2,129	1,459	1,776	1,683	1,199
100	1,771	1,100	4,701	2,885	2,861	2,214	2,654	2,624	1,616	2,013	1,643	1,303
125	1,787	0,953	5,167	2,955	3,111	2,108	2,892	3,100	1,742	2,211	1,661	1,402
150	1,836	0,866	5,511	2,978	3,287	1,975	3,001	3,438	1,790	2,280	1,676	1,508
175	1,779	0,765	5,375	2,759	3,338	1,829	3,020	3,608	1,876	2,392	1,610	1,509
200	1,718	0,670	5,889	2,919	3,629	1,819	3,429	4,353	2,113	2,713	1,623	1,604
225	1,695	0,619	6,162	2,911	3,759	1,752	3,636	4,705	2,218	2,831	1,639	1,662
250	1,715	0,591	6,343	2,919	3,934	1,727	3,699	4,939	2,294	2,922	1,612	1,690
275	1,776	0,605	6,437	2,953	4,038	1,745	3,624	4,883	2,273	2,885	1,594	1,693
300	1,693	0,513	6,569	2,902	4,089	1,605	3,880	5,660	2,415	3,131	1,606	1,808

Πίνακας 6.49 - Κανονικοποιημενα αποτελέσματα για τη έβδομη κλάση δεδομένων

n	A1		A2		A3		A4	
	niter	cpu	niter	cpu	niter	cpu	niter	cpu
50	4,12	2,90	2,27	1,67	1,00	1,00	1,77	1,10
75	4,53	2,93	2,46	2,13	1,00	1,00	1,68	1,20
100	4,70	2,89	2,65	2,62	1,00	1,00	1,64	1,30
125	5,17	2,96	2,89	3,10	1,00	1,00	1,66	1,40
150	5,51	2,98	3,00	3,44	1,00	1,00	1,68	1,51
175	5,37	2,76	3,02	3,61	1,00	1,00	1,61	1,51
200	5,89	2,92	3,43	4,35	1,00	1,00	1,62	1,60
225	6,16	2,91	3,64	4,71	1,00	1,00	1,64	1,66
250	6,34	2,92	3,70	4,94	1,00	1,00	1,61	1,69
275	6,44	2,95	3,62	4,88	1,00	1,00	1,59	1,69
300	6,57	2,90	3,88	5,66	1,00	1,00	1,61	1,81



Εικόνα 6.30 - Κλάση 7: Το διάγραμμα των κανονικοποιημένων χρόνων



Εικόνα 6.31 - Κλάση 7: Το διάγραμμα των κανονικοποιημένων επαναλήψεων

6.4.3 Βασικά Συμπεράσματα Υπολογιστικής Μελέτης

Το βασικό συμπέρασμα που μπορεί να προκύψει από την εκτενέστατη υπολογιστική μελέτη που πραγματοποιήθηκε είναι ότι ο πιο αποδοτικός αλγόριθμος από αυτούς που παρουσιάσθηκαν είναι ο αλγόριθμος Παπαρρίζου με ξεκίνημα το δάσος AKP. Στη συνέχεια, η δεύτερη θέση καταλαμβάνεται από τη τον αλγόριθμο Παπαρρίζου με απλό ξεκίνημα. Τις δύο τελευταίες θέσεις καταλαμβάνουν οι αλγόριθμοι Simplex και Balinski, οι οποίοι εναλλάσσονται, με τον δεύτερο να επικρατεί τις περισσότερες φορές **[PPS]**.

6.5 ΑΝΑΦΟΡΕΣ

[Sam] Samaras Nikolaos, (2001*), "Computational Improvements and Efficient Implementation of Two Path Pivoting Algorithms"*, PhD Dissertation, Dept of Applied Informatics, University of Macedonia.

[P1] H. Achatz, P. Kleinschmidt and K. Paparrizos, "*A dual forest algorithm for the assignment problem,*" Dimacs, vol. 4, pp. 1-10, 1990.

[PPS] C. Papamanthou, K. Paparrizos, N. Samaras, *"A Comparative Computational Study of Exterior Point Algorithms for Dense Assignment Problems"* in preparation

[SS] Stephanides G., Samaras N., *"Computational Methods with Matlab"*, Zygos Publications, Thessaloniki 1999, in Greek

ΚΕΦΑΛΑΙΟ 7

ΟΠΤΙΚΟΠΟΙΗΣΗ ΑΛΓΟΡΙΘΜΩΝ

7.1 ΕΙΣΑΓΩΓΗ

Είναι γεγονός ότι η οπτικοποίηση μέσω λογισμικού είναι ένα πάρα πολύ σημαντικό εργαλείο για την παρουσίαση και εμπέδωση αλγορίθμων. Με τον όρο «οπτικοποίηση μέσω λογισμικού» εννοούμε τη χρήση γραφικών και animation έτσι ώστε να παρουσιασθούν αναλυτικά τα βήματα ενός αλγορίθμου. Ειδικότερα σε μαθήματα Αλγορίθμων, η οπτικοποίηση αποτελεί μια από τις πιο διαδομένες μεθόδους διδασκαλίας και ανήκει σε ένα νέο επιστημονικό πεδίο έρευνας.

Χρησιμοποιώντας λογισμικό οπτικοποίησης, στοχεύουμε να παρουσιάσουμε την κατάσταση του προβλήματος σε κάθε επανάληψη του αλγορίθμου. Αυτή η διαδικασία βοηθάει κάποιον να κατανοήσει τις ενέργειες που πρέπει να εκτελεσθούν έτσι ώστε να μεταβούμε από μία επανάληψη u στην επανάληψη $u+1$. Ωστόσο, κάποιος δε πρόκειται να μάθει τη λειτουργία του αλγορίθμου κάνοντας χρήση ενός λογισμικού οπτικοποίησης **[BCS]**. Ο αλγόριθμος θα πρέπει να μελετηθεί διεξοδικά πριν εκτελεσθεί η οπτικοποίηση, ιδιαίτερα όταν μελετώνται αλγόριθμοι μεγάλης πολυπλοκότητας όπως είναι οι αλγόριθμοι Δικτυακού Προγραμματισμού.

Όλοι οι αλγόριθμοι που παρουσιάσθηκαν προγραμματίστηκαν σε Java έτσι ώστε να επιτευχθεί μια γραφική παρουσίαση των κυριότερων λειτουργιών των αλγορίθμων. Δε χρησιμοποιήθηκε ιδιαίτερη αντικειμενοστραφής προσέγγιση άλλα οι προσπάθειες εστιάστηκαν στο τελικό οπτικό αποτέλεσμα. Όλες οι μικροεφαρμογές Java (Java Applets) είναι διαθέσιμες προς εκτέλεση στο web.

Η ανάπτυξη των μικροεφαρμογών είχε κατά κύριο λόγο εκπαιδευτικό χαρακτήρα και έγινε με γνώμονα τα παρακάτω σημεία**[PP]**:

- Οι φοιτητές πρέπει να αποκτήσουν μια γενική εικόνα της λειτουργίας του αλγορίθμου.
- Πρέπει να γίνει ξεκάθαρο με ποιο τρόπο το δέντρο κάθε επανάληψης αλληλεπιδρά με την τρέχουσα κατάσταση των μεταβλητών του αλγορίθμου. Επίσης, οι φοιτητές πρέπει να καταλάβουν πως επιτυγχάνεται η ανανέωση του τρέχοντος δέντρου.
- Ο διδάσκων θα πρέπει να δώσει έμφαση στον τρόπο που χρησιμοποιείται το δέντρο από το λογισμικό. Με αυτόν τον τρόπο, ο διδάσκων δε θα χρειαστεί να ζωγραφίζει τις διαδοχικές καταστάσεις του δέντρου στον πίνακα.
- Η χρήση της τεχνολογίας και μεθόδων βασισμένων σε υπολογιστές κάνει τα μαθήματα Αλγορίθμων πιο ευχάριστα και πιο ενδιαφέροντα.

Στη συνέχεια θα αναφερόμαστε στους αλγορίθμους που παρουσιάσθηκαν στα προηγούμενα κεφάλαια με τις συντομογραφίες που αναφέρθηκαν και στο κεφάλαιο 6 :

Πίνακας 7.1 – Συντομογραφίες Ονομάτων Αλγορίθμων

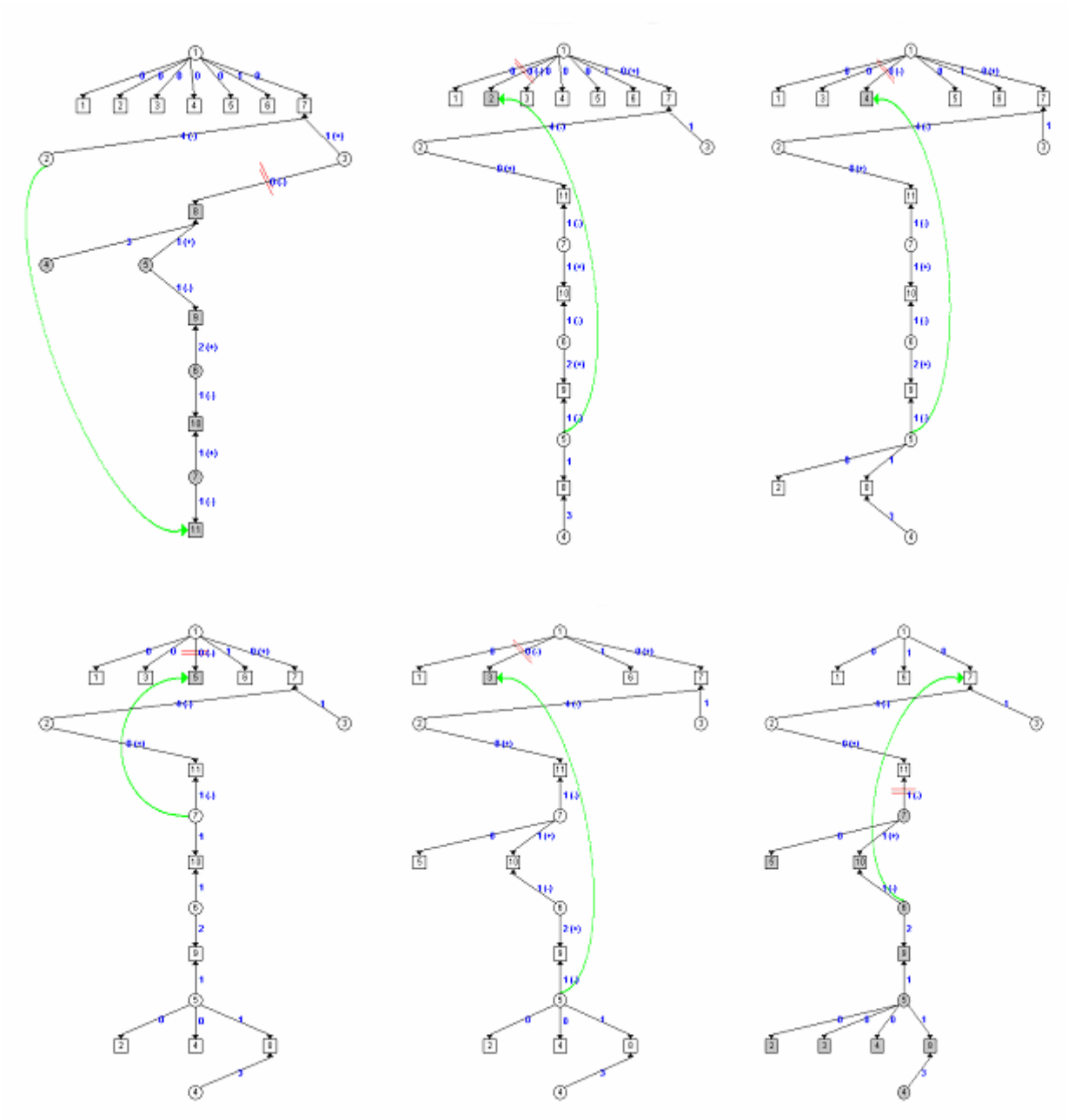
Ακρωνύ-μιο	**Αλγόριθμος**
T1	Πρωτεύων Αλγόριθμος Simplex για το πρόβλημα Μεταφοράς
A1	Πρωτεύων Αλγόριθμος Simplex για το πρόβλημα Αντιστοίχησης
T2	Αλγόριθμος Παπαρρίζου με ξεκίνημα το δέντρο Balinski για το πρόβλημα Μεταφοράς
A2	Αλγόριθμος Παπαρρίζου με ξεκίνημα το δέντρο Balinski για το πρόβλημα Αντιστοίχησης
T3	Αλγόριθμος Παπαρρίζου με ξεκίνημα το δάσος ΑΚΡ για το πρόβλημα Μεταφοράς
A3	Αλγόριθμος Παπαρρίζου με ξεκίνημα το δάσος ΑΚΡ για το πρόβλημα Αντιστοίχησης
T4	Αλγόριθμος Παπαρρίζου με απλό ξεκίνημα για το πρόβλημα Μεταφοράς
A4	Αλγόριθμος Παπαρρίζου με απλό ξεκίνημα για το πρόβλημα Αντιστοίχησης

7.2 ΠΑΡΟΥΣΙΑΣΗ ΤΟΥ ΛΟΓΙΣΜΙΚΟΥ

Στη συνέχεια θα παρουσιάσουμε τα frames που παράγει το λογισμικό όταν εκτελείται για τους αλγόριθμους που παρουσιάσθηκαν. Στις υποενότητες που ακολουθούν θα παρουσιάσουμε τις μερικές διαδοχικές επαναλήψεις όλων των αλγορίθμων για τυχαία δεδομένα όπως αυτές εκτελούνται από το λογισμικό**[PPS]**.

7.2.1 Οπτικοποίηση Αλγορίθμου T1

Παρακάτω παρουσιάζουμε τις πρώτες 6 επαναλήψεις του αλγορίθμου για την επίλυση ενός *7x11* προβλήματος Μεταφοράς.



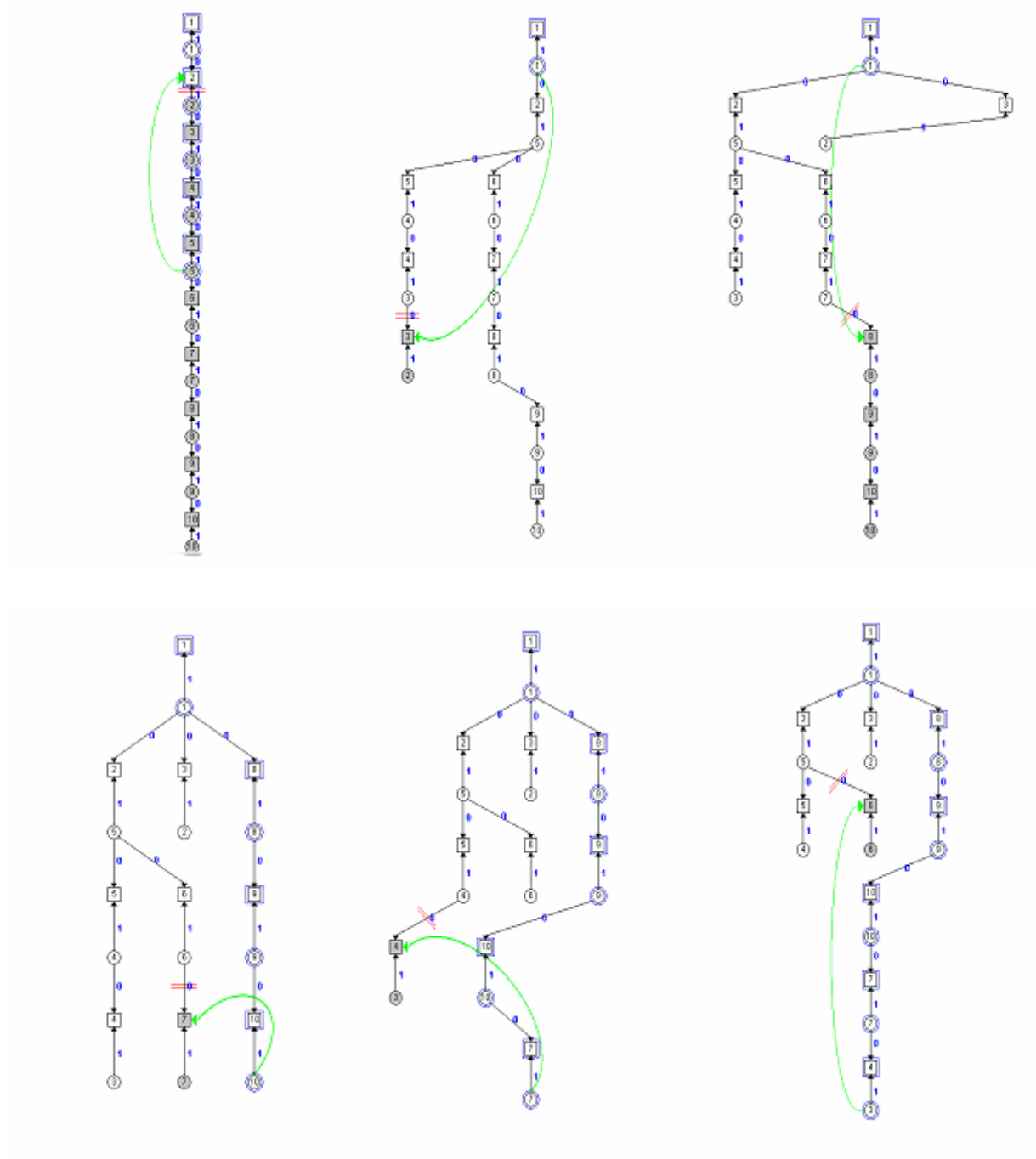
Εικόνα 7.0 – Οπτικοποίηση του αλγορίθμου για ένα 7x11 πρόβλημα Μεταφοράς

Παρατηρείστε ότι σημειώνονται τα τόξα που ανήκουν στα σύνολα C^+ και C^-, το εισερχόμενο και το εξερχόμενο τόξο, έτσι ώστε ο χρήστης να μπορεί να διακρίνει τις βασικές

μεταβλητές του αλγορίθμου. Επίσης, οι κόμβοι που ανήκουν στο δέντρο $T*$ ζωγραφίζονται με γκρι χρώμα.

7.2.2 Οπτικοποίηση Αλγόριθμου A1

Παρακάτω παρουσιάζουμε τις πρώτες 6 επαναλήψεις του αλγορίθμου για την επίλυση ενός προβλήματος Αντιστοίχησης μεγέθους 10.



Εικόνα 7.1 – Οπτικοποίηση του αλγορίθμου για ένα πρόβλημα Αντιστοίχησης μεγέθους 10

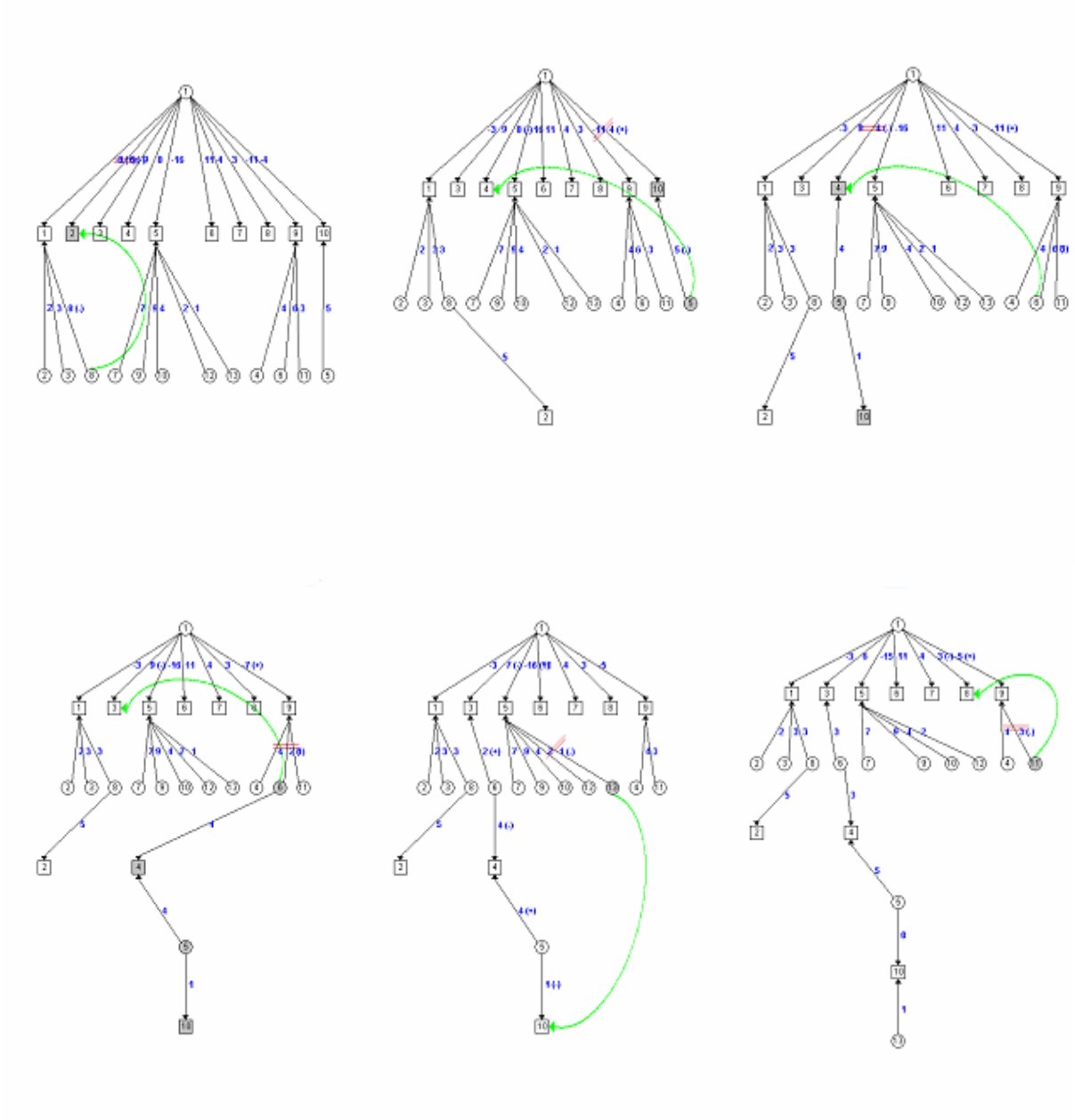
Παρατηρείστε ότι στην πρώτη εικόνα, το αρχικό εφικτό δέντρο είναι ένας δρόμος(*path*) από τον κόμβο στήλη 1 προς τον κόμβο γραμμή 10, όπως ακριβώς αναφέρθηκε και στην περιγραφή του αλγορίθμου. Επίσης, για την καλύτερη κατανόηση του αλγορίθμου, βλέπουμε ότι οι κόμβοι που ανήκουν στον δρόμο $P[T, g]$ περιβάλλονται με χρώμα μπλε, έτσι ώστε ο

χρήστης να μπορεί να διαπιστώσει αν $h \in P[T, g]$ και να καταλάβει ποιο θα είναι το εξερχόμενο τόξο.

Το εξερχόμενο τόξο, όπως και στο πρόβλημα Μεταφοράς, μαρκάρεται με δύο παράλληλες κόκκινες γραμμές, ενώ το εισερχόμενο τόξο είναι πράσινο ημιελλειψοειδές.

7.2.3 Οπτικοποίηση Αλγορίθμου Τ2

Το δέντρο που αντιστοιχεί στην πρώτη εικόνα είναι το δέντρο Balinski όπως αυτό περιγράφηκε για το πρόβλημα Μεταφοράς. Παρατηρείστε ότι όλοι οι κόμβοι γραμμές βρίσκονται στο επίπεδο 2 εκτός από τη ρίζα του δέντρου.

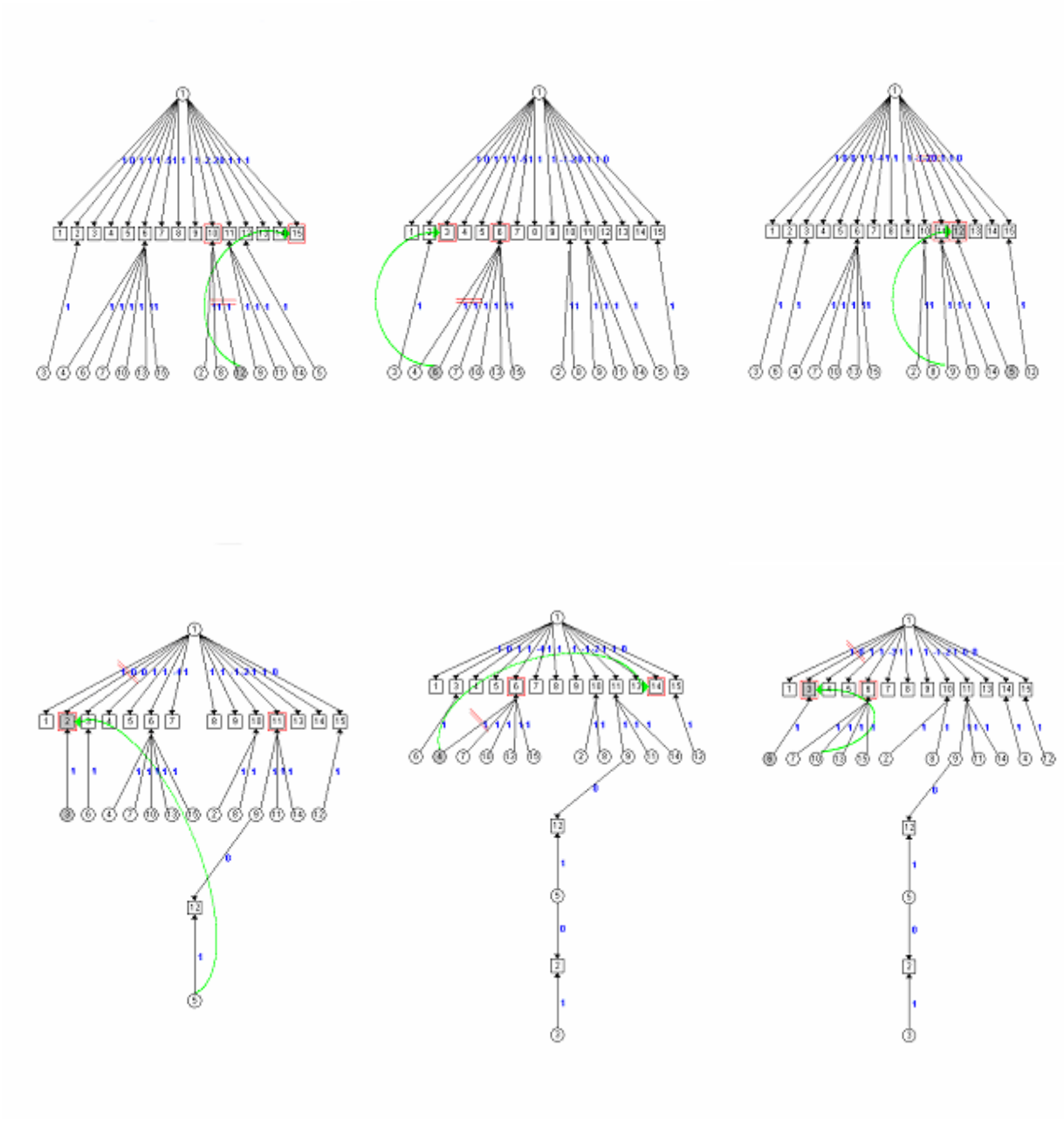


Εικόνα 7.2 – Οπτικοποίηση του αλγορίθμου για ένα 13x10 πρόβλημα Μεταφοράς

Βλέπουμε ότι στο δέντρο Balinski υπάρχουν τόξα $(1, j)$ τέτοια ώστε $x_{1j} < 0$ πράγμα που σημαίνει ότι $F \neq \varnothing$. Αυτό άλλωστε θα φανεί και στην επόμενη οπτικοποιήση που θα παρουσιασθεί ο ίδιος αλγόριθμος για το πρόβλημα Αντιστοίχησης.

7.2.4 Οπτικοποίηση Αλγορίθμου Α2

Παρακάτω παρουσιάζουμε τις πρώτες 6 επαναλήψεις του αλγορίθμου για την επίλυση ενός προβλήματος Αντιστοίχησης μεγέθους 15.



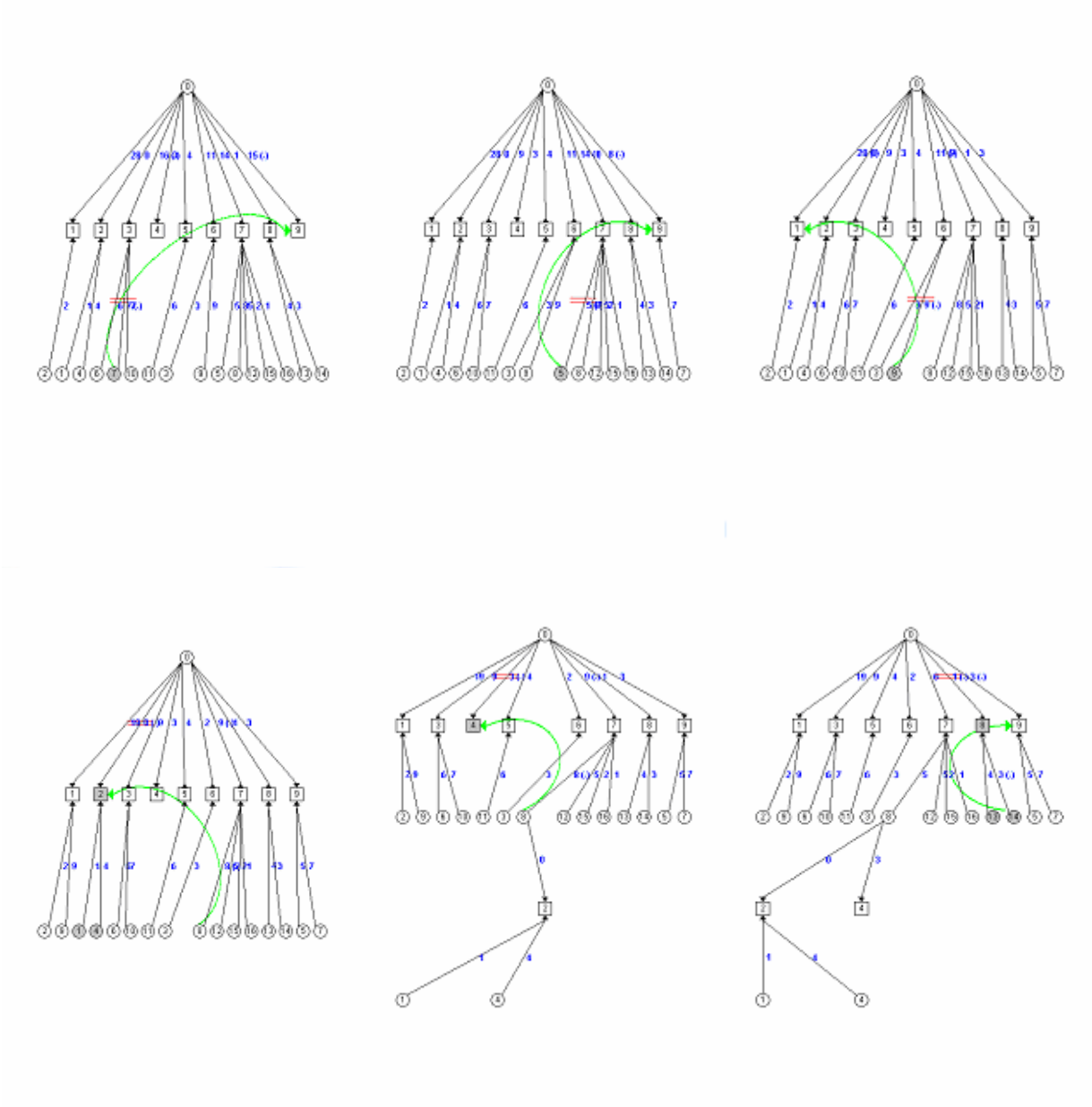
Εικόνα 7.3 – Οπτικοποίηση του αλγορίθμου πρόβλημα Αντιστοίχησης μεγέθους 15

Αυτή η οπτικοποίηση έχει ακόμη ένα ιδιαίτερο χαρακτηριστικό. Παρατηρούμε ότι δύο κόμβοι στο δέντρο περιβάλλονται σε κάθε επανάληψη με κόκκινο χρώμα. Αυτοί είναι οι κόμβοι h και f, οι βαθμοί των οποίων χρησιμοποιούνται για τον καθορισμό του εξερχόμενου τόξου (k,ℓ).

Οι τεχνικές σχεδίασης που χρησιμοποιούνται και εδώ είναι όμοιες με αυτές που χρησιμοποιήθηκαν και από τους άλλους αλγορίθμους. Δίπλα σε κάθε τόξο (i,j) αναγράφονται οι τιμές των μεταβλητών απόφασης $x_{ij} \in \{0,1\}$. Επίσης, οι κόμβοι που ζωγραφίζονται με χρώμα γκρι ανήκουν στο δέντρο T^*. Ρίζα του δέντρου παραμένει πάντα ο κόμβος-γραμμή 1.

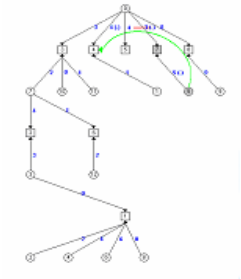
7.2.5 Οπτικοποίηση Αλγορίθμου Τ3

Στη συνέχεια, θα παρουσιάσουμε την επίλυση μέσω λογισμικού ενός *16x9* προβλήματος Μεταφοράς μέσω του λογισμικού και χρησιμοποιώντας τον αλγόριθμο Τ3.Παρατηρείστε ότι τα τόξα $(0, j)$ ζωγραφίζονται με διαφορετικό τρόπο από τα τόξα $(j, 0)$ για να μπορούμε να δια-



Εικόνα 7.4 – Οπτικοποίηση του αλγορίθμου για ένα 16x9 πρόβλημα Μεταφοράς

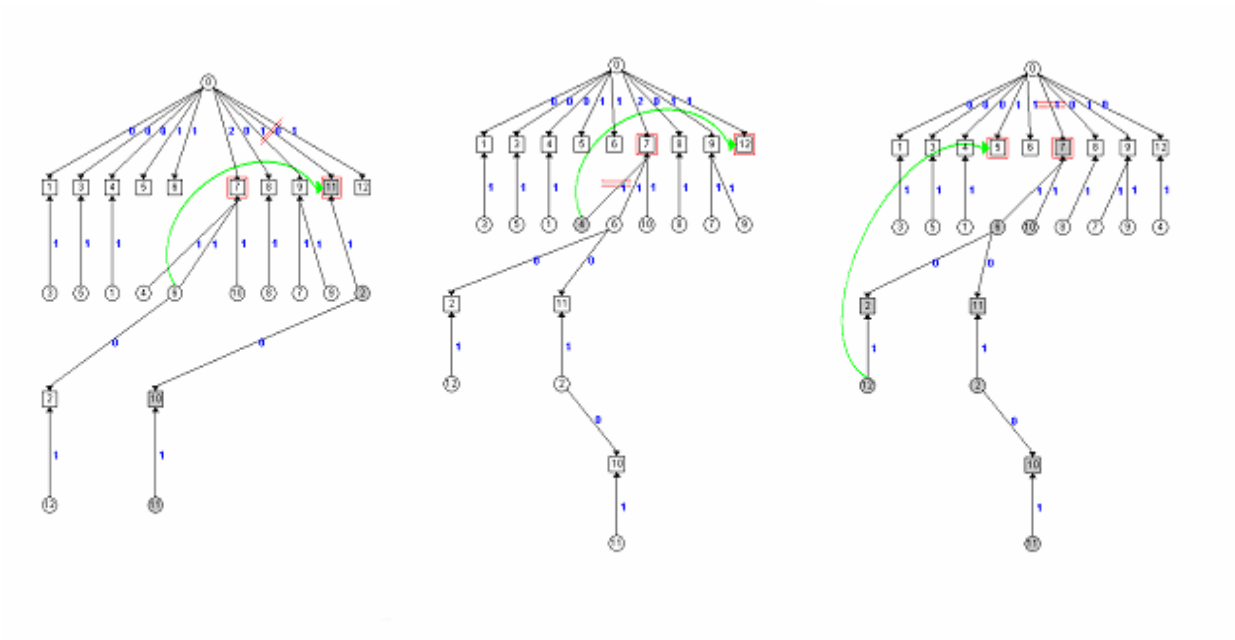
κρίνουμε ποιοι κόμβοι ανήκουν στο F^S και ποιοι ανήκουν στο F^D.



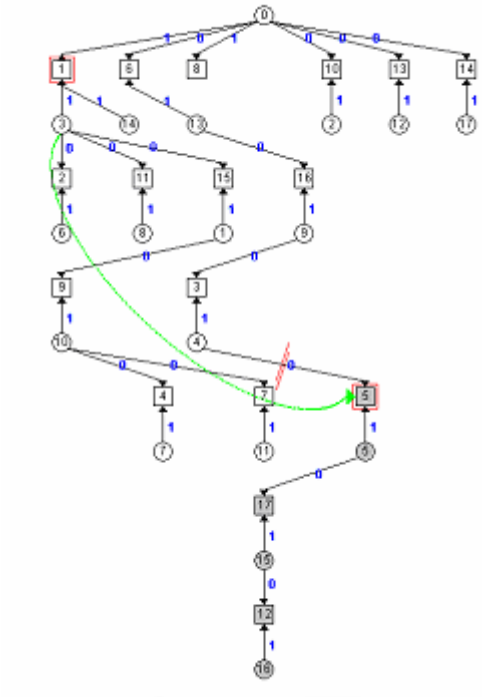
Εικόνα 7.5 – Επιλύνοντας ένα 12x8 πρόβλημα Μεταφοράς

7.2.6 Οπτικοποίηση Αλγορίθμου A3

Παρακάτω παρουσιάζουμε 3 διαδοχικές επαναλήψεις του αλγορίθμου για την επίλυση ενός προβλήματος Αντιστοίχησης μεγέθους 15.



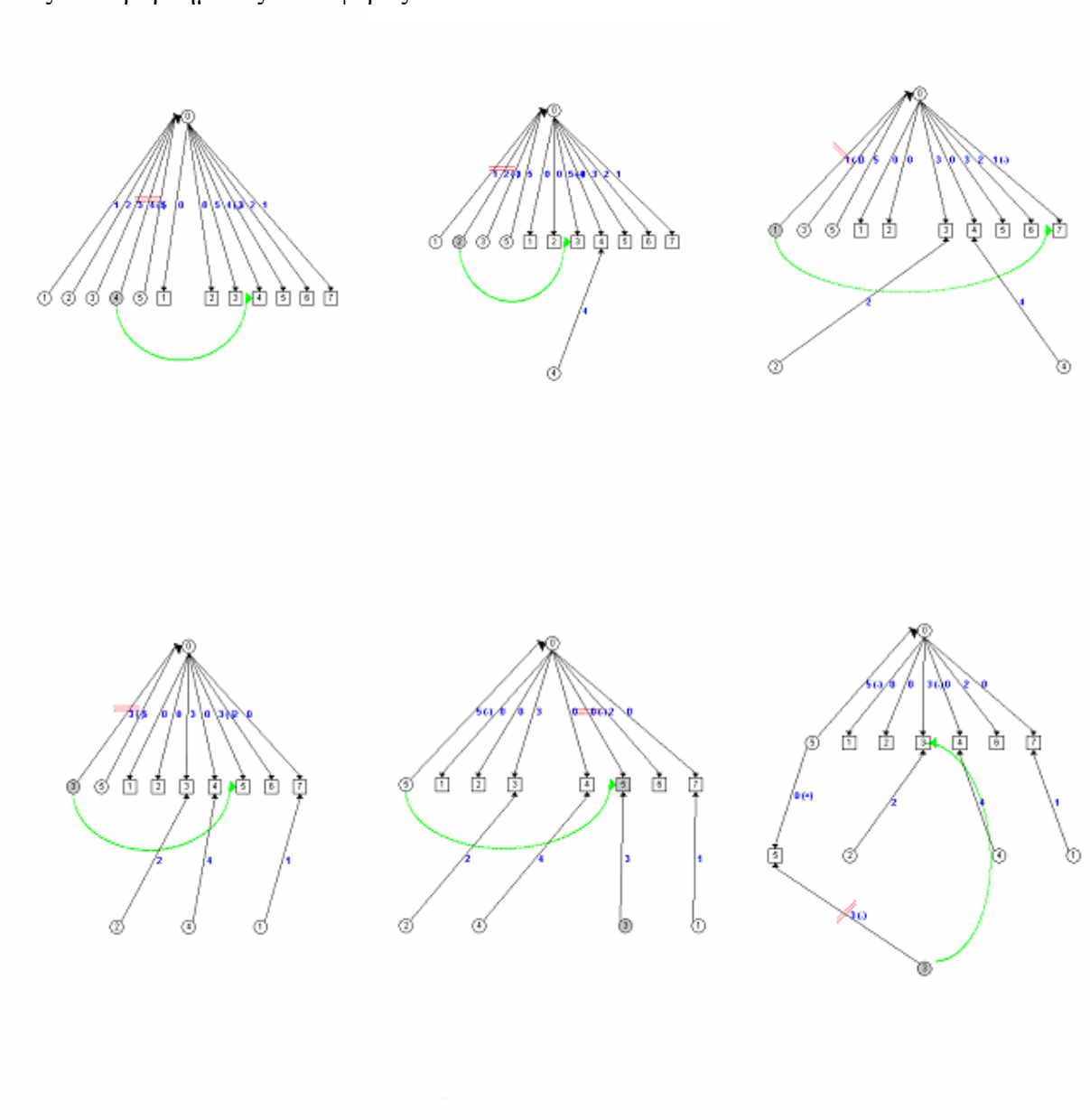
Εικόνα 7.6 – Επιλύνοντας ένα πρόβλημα Αντιστοίχησης μεγέθους 12



Εικόνα 7.7 - Επιλύνοντας ένα πρόβλημα Αντιστοίχησης μεγέθους 17

7.2.7 Οπτικοποίηση Αλγορίθμου Τ4

Παρακάτω παρουσιάζουμε τις έξι πρώτες επαναλήψεις του αλγορίθμου για την επίλυση ενός *5x7* προβλήματος Μεταφοράς.

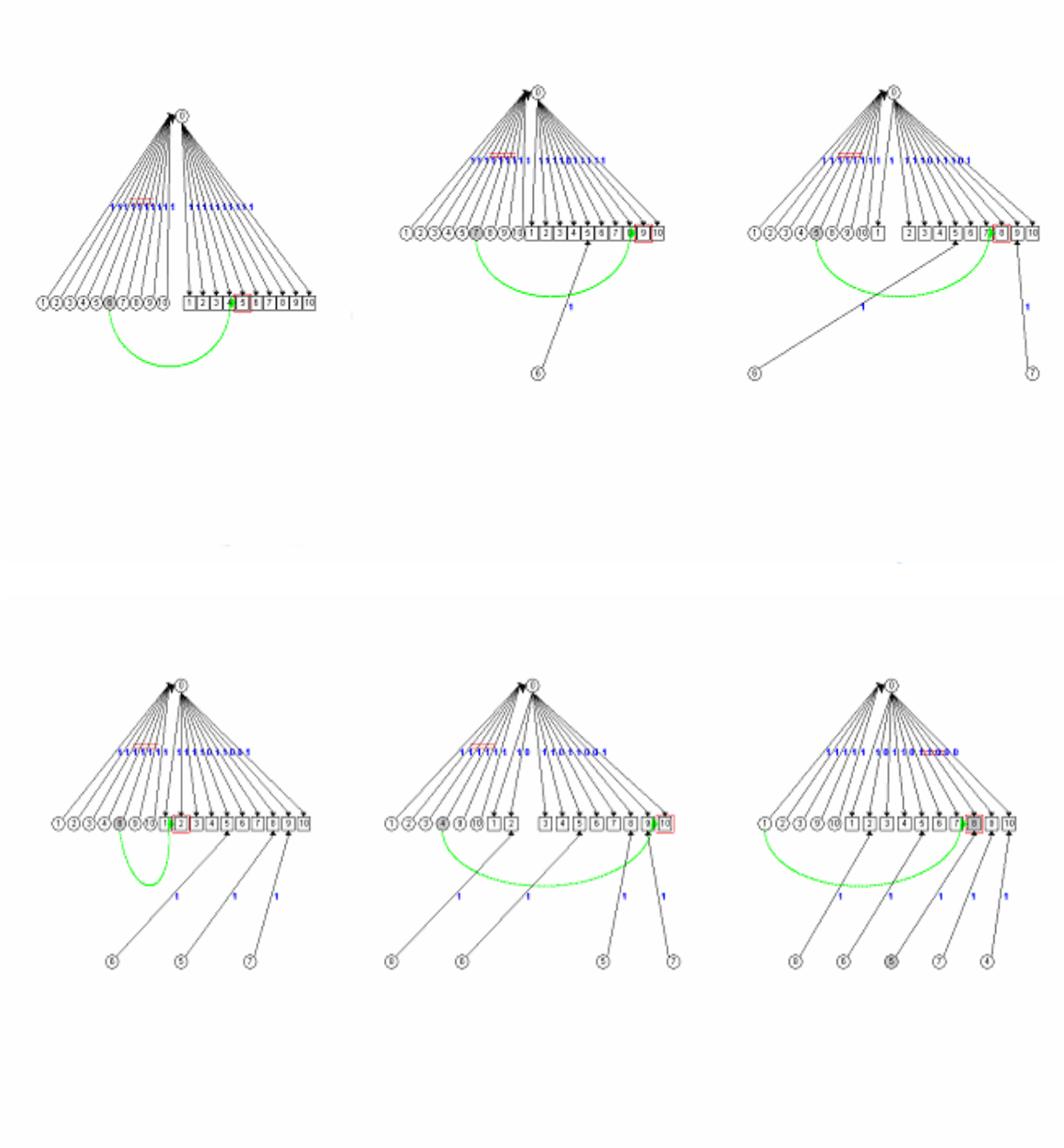


Εικόνα 7.8 – Οι πρώτες 6 επαναλήψεις του αλγορίθμου Τ4

Παρατηρείστε ότι αρχικά όλοι οι κόμβοι βρίσκονται σε βάθος 1. Το σύνολο F^S αποτελείται από τους κόμβους γραμμές(πρώτη εικόνα) ενώ το σύνολο F^D αποτελείται από τους κόμβους στήλες. Οι επόμενες 3 επαναλήψεις του αλγορίθμου φαίνονται στο παρακάτω σχήμα.

7.2.8 Οπτικοποίηση Αλγορίθμου Α4

Τέλος, θα παρουσιάσουμε την οπτικοποίηση του αλγορίθμου Α4. Παρουσιάζονται ενδεικτικά οι 6 πρώτες επαναλήψεις για ένα πρόβλημα Αντιστοίχησης μεγέθους 10. Παρατηρείστε ότι επειδή εδώ μας ενδιαφέρει μόνο ο βαθμός του κόμβου h για την επιλογή του εξερχόμενου τόξου, μόνο αυτός περιβάλλεται με κόκκινο.



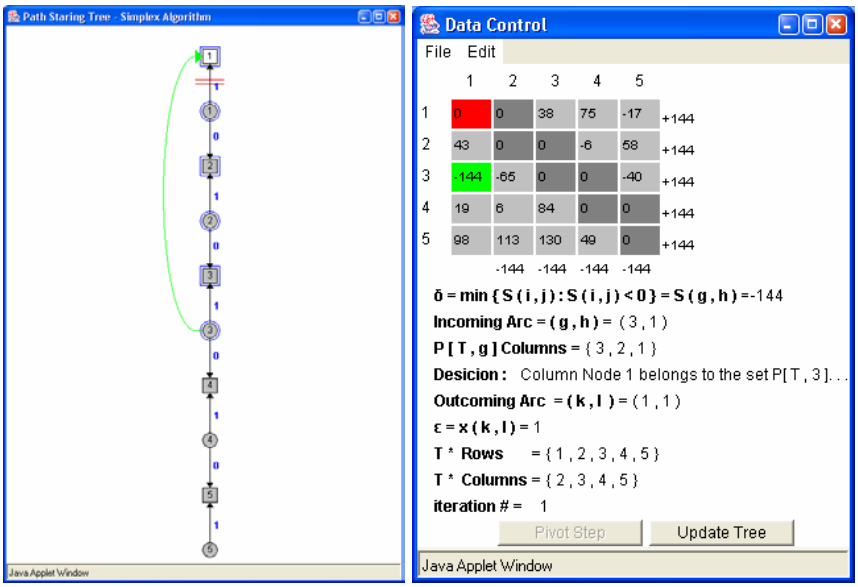
Εικόνα 7.9 – Οι πρώτες 6 επαναλήψεις του αλγορίθμου Α4

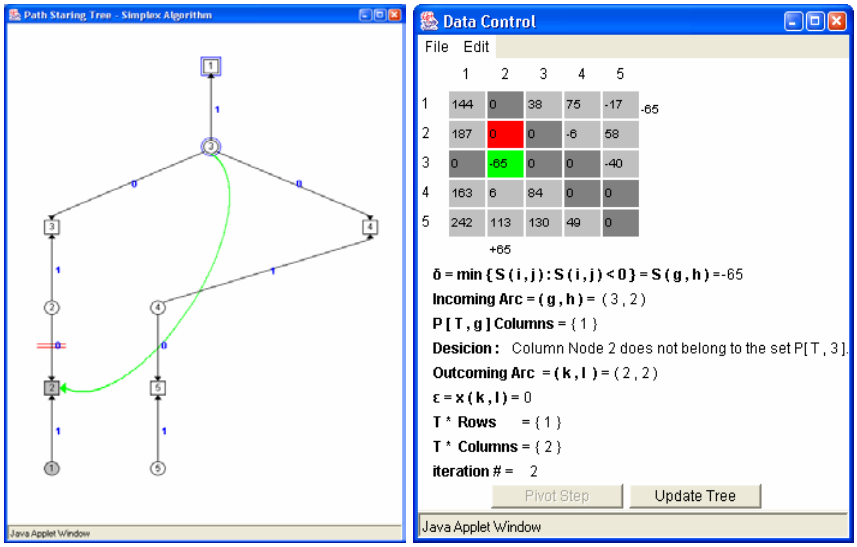
7.3 Η ΦΟΡΜΑ ΕΛΕΓΧΟΥ ΔΕΔΟΜΕΝΩΝ

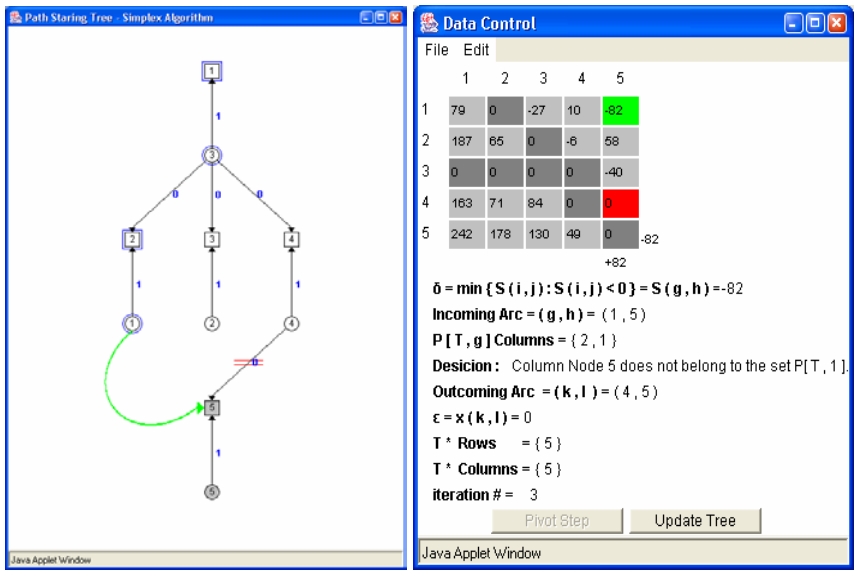
Επειδή το λογισμικό έχει εκπαιδευτικό χαρακτήρα, τα frames που παρουσιάσαμε θα πρέπει να συνοδεύονται και από έναν τρόπο αναπαράστασης των δεδομένων που αντιπροσωπεύουν. Έτσι, προσθέσαμε στο λογισμικό και μία επιπλέον φόρμα όπου αναγράφονται οι τιμές των διάφορων κρίσιμων μεταβλητών του αλγορίθμου. Σε αυτήν τη φόρμα, η οποία είναι διαφορετική για κάθε αλγόριθμο, αναγράφονται οι τιμές μεταβλητών όπως το εισερχόμενο τόξο (g,h), το εξερχόμενο τόξο (k,ℓ), ο πίνακας των μειωμένων κοστών s_{ij}, καθώς και άλλες μεταβλητές. Η ανανέωση του δέντρου συμβαδίζει πάντα με την ανανέωση της φόρμας και έτσι πετυχαίνεται πάντα μια τέλεια οπτικοποιήση του αλγορίθμου.

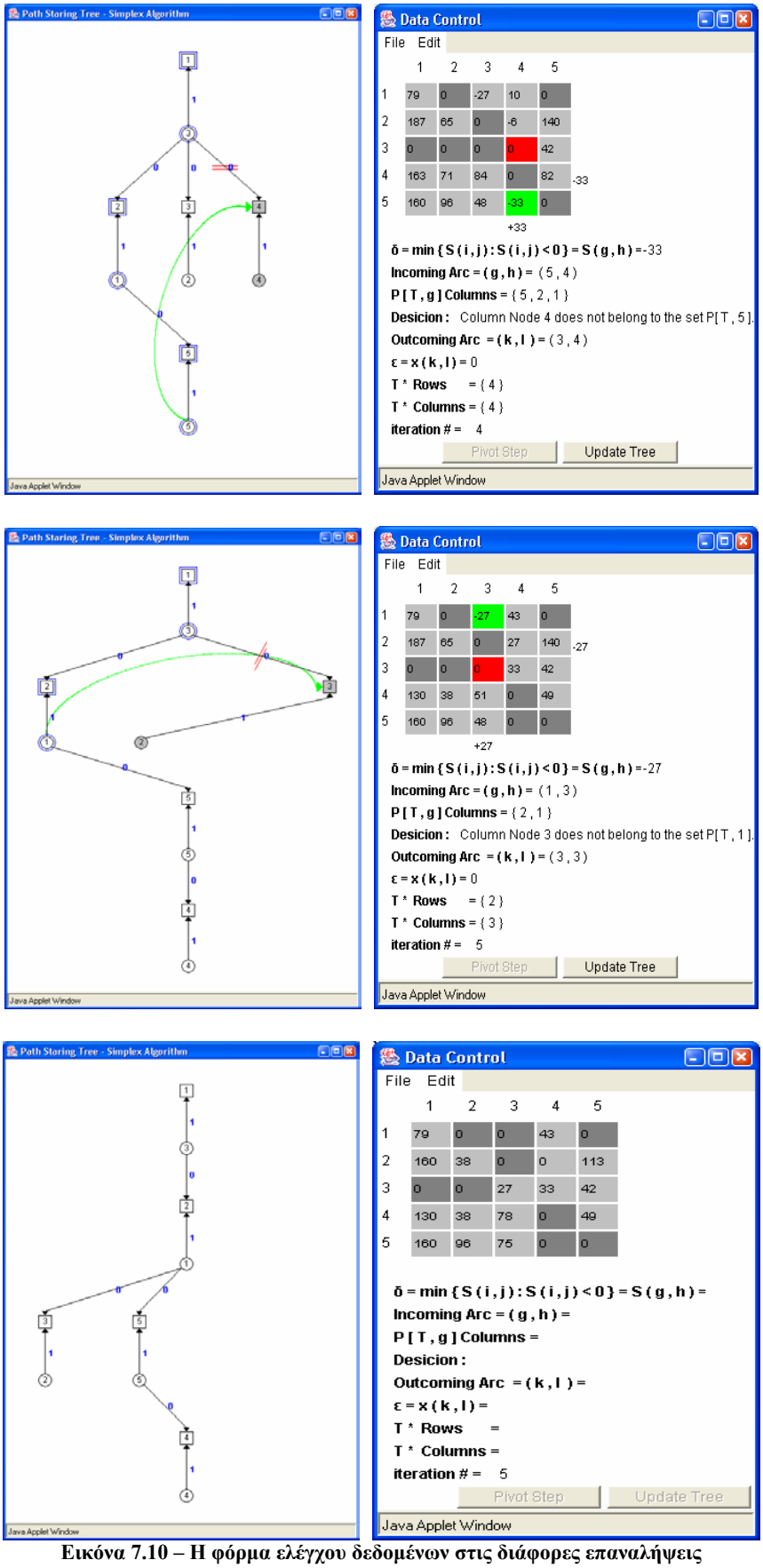
Ελέγχοντας με προσοχή τη φόρμα, ο χρήστης θα μπορεί να διαπιστώνει πως ανανεώνονται οι μεταβλητές του αλγορίθμου. Παράλληλα, επάνω στη φόρμα βρίσκονται και τα κύρια χειριστήρια για την εκτέλεση του αλγορίθμου. Η επίλυση ενός προβλήματος

Αντιστοίχησης μεγέθους 5, χρησιμοποιώντας τον αλγόριθμο A1 μέσω του λογισμικού φαίνεται καθαρά στις παρακάτω εικόνες:







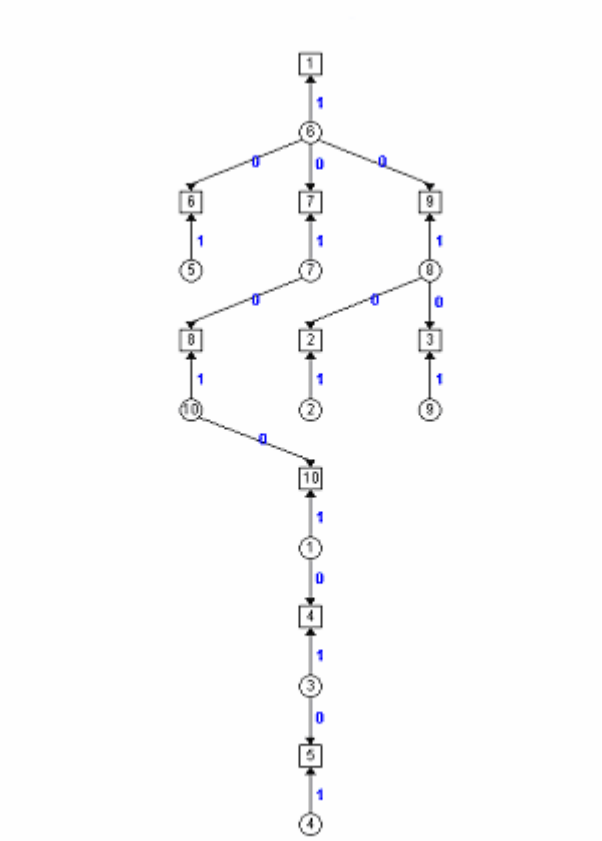


Εικόνα 7.10 – Η φόρμα ελέγχου δεδομένων στις διάφορες επαναλήψεις

Παρατηρείστε ότι το πράσινο κελί του πίνακα s_{ij} αντιστοιχεί στο εισερχόμενο τόξο, ενώ το κόκκινο κελί του πίνακα αντιστοιχεί πάντα στο εξερχόμενο τόξο. Παράλληλα, δίπλα από κάθε γραμμή και στήλη που ανήκουν στο αποκομμένο δέντρο T^*, αναγράφονται οι τιμές που πρέπει να προστεθούν και αν αφαιρεθούν έτσι ώστε να επιτευχθεί ανανέωση του πίνακα των μειωμένων κοστών. Με αυτόν τον τρόπο ο χρήστης μπορεί να κατανοήσει τη μέθοδο ανανέωσης των δεδομένων και να την εφαρμόσει και αυτός για να λύσει ένα πρόβλημα στο χαρτί. Τέλος, παρατηρείστε ότι το δέντρο που παράγεται μετά από 6 επαναλήψεις(εικόνα 7.10) είναι βέλτιστο.

7.4 Η ΤΕΧΝΙΚΗ ΣΧΕΔΙΑΣΗΣ

Κλείνοντας το κεφάλαιο της οπτικοποιήσης, θα αναφερθούμε περιληπτικά στη προγραμματιστική μέθοδο που χρησιμοποιήσαμε για την απεικόνιση στο επίπεδο των δέντρων που χρησιμοποιούν οι αλγόριθμοι. Έστω λοιπόν ένα δέντρο T που κατασκευάζεται από το λογισμικό όπως αυτό της εικόνας 7.11.



Εικόνα 7.11 – Ένα δέντρο *T*

Παρατηρείστε ότι ο αλγόριθμος σχεδίασης κάνει βέλτιστη χρησιμοποίηση του χώρου**[PP]**. Χρησιμοποιείται το βάθος του δέντρου για να αποφασισθεί σε πόσα επίπεδα θα χωρισθεί το πλαίσιο σχεδίασης. Σε κάθε επίπεδο, υπάρχουν μόνο κόμβοι που ανήκουν στο ίδιο βάθος. Αυτός είναι ο λόγος που η απόσταση μεταξύ κόμβων που ανήκουν σε διαδοχικά επίπεδα εξαρτάται από το μέγεθος του προβλήματος που επιλύουμε. Συγκεκριμένα, οι κάθετες αποστάσεις μεταξύ των κόμβων είναι αντιστρόφως ανάλογες με το μέγεθος του προβλήματος. Θεωρητικά, μπορούμε να χρησιμοποιήσουμε αυτήν τη τεχνική για να απεικονίσουμε δέντρα προβλημάτων οποιουδήποτε μεγέθους. Ωστόσο, το μέγεθος περιορίζεται στον ακέραιο 20, για να έχουμε ένα καλό αισθητικό αποτέλεσμα(αυτό σημαίνει ότι με το λογισμικό οπτικοποιήσης

μπορούμε να λύσουμε προβλήματα Αντιστοίχησης μεγέθους $n \le 20$ και προβλήματα Μεταφοράς μεγέθους $m \times n$ με $m, n \le 20$).

Συγκεκριμένα, ο αλγόριθμος σχεδίασης που χρησιμοποιήσαμε λειτουργεί ως εξής. Όπως και στους αλγορίθμους που περιγράψαμε στα προηγούμενα κεφάλαια, έτσι και στην οπτικοποίηση χρησιμοποιείται και ανανεώνεται το διάνυσμα βάθους d. Υπενθυμίζουμε ότι στο διάνυσμα αυτό αποθηκεύονται τα βάθη των κόμβων του δέντρου έτσι ώστε το στοιχείο του διανύσματος $d(i)$ να ισούται με το βάθος του κόμβου i. Το διάνυσμα d που αντιστοιχεί στο δέντρο της εικόνας 7.11 θα είναι συνεπώς:

$$d = \begin{bmatrix} 7 & 5 & 9 & 11 & 3 & 1 & 3 & 3 & 5 & 5 & 0 & 4 & 4 & 8 & 10 & 2 & 2 & 4 & 2 & 6 \end{bmatrix}$$

Παρατηρείστε ότι το βάθος του κόμβου στήλη i αποθηκεύεται στη θέση $i+n$ του διανύσματος d, όταν πρόκειται για πρόβλημα Αντιστοίχησης μεγέθους n, ενώ αποθηκεύεται στη θέση $i+m$ όταν πρόκειται για ένα $m \times n$ πρόβλημα Μεταφοράς. Ο αλγόριθμος σχεδίασης χρησιμοποιεί έναν $q \times r$ πίνακα A τέτοιον ώστε:

$$q = \max\{d(i),\ i = 1..e\} + 1$$

όπου e το άθροισμα των κόμβων του προβλήματος(είναι $e = 2n$ ή $e = m + n$, ανάλογα εάν έχουμε πρόβλημα Αντιστοίχησης ή Μεταφοράς αντίστοιχα). Επίσης

$$r = \max\{b(i),\ i = 0..q-1\}$$

όπου b ένα διάνυσμα q θέσεων, στη θέση $b(i)$ του οποίου αποθηκεύεται το πλήθος των κόμβων στο i επίπεδο. Οι γραμμές του πίνακα A αριθμούνται από το 0 έως $m-1$ έτσι ώστε να αναπαριστούν τα αντίστοιχα βάθη των κόμβων που ανήκουν στη συγκεκριμένη γραμμή. Ο πίνακας A κατασκευάζεται σε κάθε επανάληψη του αλγορίθμου με έναν τρόπο που απεικονίζει τη «γεωμετρική φύση» του δέντρου. Αυτό μπορεί εύκολα να γίνει κατανοητό με ένα παράδειγμα. Έτσι, ο πίνακας $A(T)$ που αντιπροσωπεύει το δέντρο της εικόνας 7.11 θα είναι ο παρακάτω 12×3 πίνακας:

$$A(T) = \begin{bmatrix} 11 & \infty & \infty \\ 6 & \infty & \infty \\ 16 & 17 & 19 \\ 5 & 7 & 8 \\ 18 & 12 & 13 \\ 10 & 2 & 9 \\ 20 & \infty & \infty \\ 1 & \infty & \infty \\ 14 & \infty & \infty \\ 3 & \infty & \infty \\ 15 & \infty & \infty \\ 4 & \infty & \infty \end{bmatrix}$$

Με τον τρόπο που περιγράψαμε είναι εύκολο να αναθέσουμε σε κάθε κόμβο του δέντρου ένα χώρο που περιορίζεται από ένα ορθογώνιο μεταβλητών διαστάσεων, οι οποίες εξαρτώνται κάθε φορά από το μέγεθος του προβλήματος που λύνουμε, το πλήθος των κόμβων που ανήκουν στο ίδιο επίπεδο καθώς και το διαθέσιμο χώρο σχεδίασης. Έτσι, υπολογίζουμε εύκολα τις συντεταγμένες κάθε κόμβου και τους απεικονίζουμε στο καρτεσιανό επίπεδο. Ο πίνακας A ανανεώνεται σε κάθε επανάληψη και καθορίζει τη σχεδίαση του νέου δέντρου. Έτσι, οι κόμβοι που ανήκουν στο ίδιο βάθος αλλάζουν τη θέση τους ανάλογα με τον αριθμό των «ισοβαθών» κόμβων. Παράλληλα, τα βασικά τόξα του δέντρου μπορούν να σχεδιαστούν εύκολα χρησιμοποιώντας τις ήδη υπολογισμένες συντεταγμένες των κόμβων του δέντρου.

Τελειώνοντας με τη λειτουργία του αλγορίθμου σχεδίασης, θα αναφερθούμε στην καμπύλη που χρησιμοποιείται για την απεικόνιση του εισερχόμενου τόξου (g,h). Αυτή η καμπύλη είναι ένα ελλειψοειδές, του οποίου τα σημεία $M(x,y)$ μπορούν να παρασταθούν στο επίπεδο με τις παραμετρικές εξισώσεις της έλλειψης οι οποίες είναι

$$x = r\sin t \wedge y = \frac{a}{2}\cos t \ , \ t \in [0, 2\pi)$$

,όπου a είναι η απόσταση μεταξύ των κόμβων g, h και r είναι μια τιμή της επιλογής μας που δείχνει την κυρτότητα της έλλειψης. Στο πρόγραμμα έχουμε θέσει $r = 80$ [1]. Οι παραπάνω εξισώσεις μπορούν να χρησιμοποιηθούν για τη σχεδίαση ελλείψεων των οποίων οι άξονες είναι παράλληλοι με τους άξονες του συστήματος συντεταγμένων(αυτό συμβαίνει στην πρώτη επανάληψη της εικόνας 7.1) και των οποίων τα κέντρα είναι η αρχή των αξόνων $O(0,0)$. Επειδή τις περισσότερες φορές η έλλειψη που πρέπει να σχεδιαστεί παρουσιάζει μια κλίση ως προς τους άξονες του συστήματος, υπολογίζουμε τη γωνία ϑ που σχηματίζεται όταν η ευθεία που ενώνει τους κόμβους g, h περιστραφεί έως ότου συναντήσει την ευθεία $y = 0$. Η γωνία αυτή υπολογίζεται εύκολα θέτοντας:

$$\vartheta = \arctan\left(\left|\frac{y_g - y_h}{x_g - x_h}\right|\right)$$

, όπου x_g, y_g είναι οι συντεταγμένες του κόμβου g και αντίστοιχα x_h, y_h είναι οι συντεταγμένες του κόμβου h. Στη συνέχεια, υπολογίζοντας και το νέο κέντρο $K(x_0, y_0)$ και εφαρμόζοντας μια περιστροφή και μια μετατόπιση (σχήμα 7.12) πετυχαίνουμε πάντα το σχεδιασμό του σωστού ελλειψοειδούς, του οποίου οι συντεταγμένες τελικά είναι **[Kl], [Pk]**:

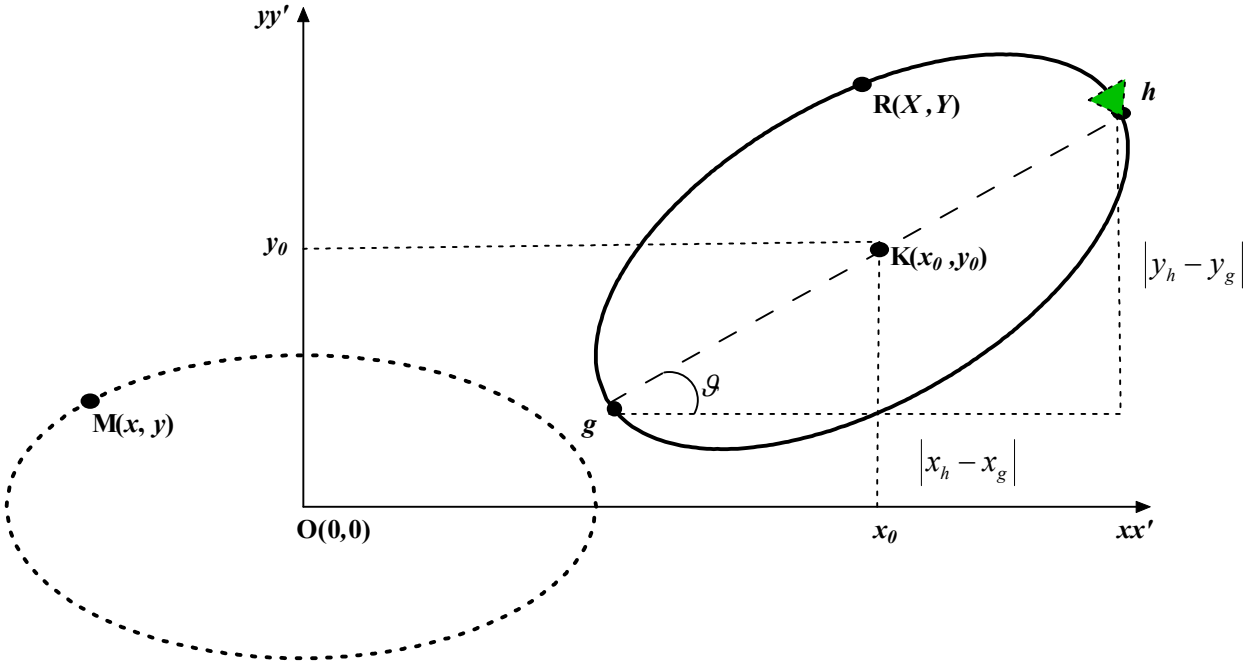
$$X = x_0 + x\cos\vartheta + y\sin\vartheta = x_0 + r\sin t\cos\vartheta + \frac{a}{2}\cos t\sin\vartheta$$

$$Y = y_0 + y\cos\vartheta - x\sin\vartheta = y_0 + \frac{a}{2}\cos t\cos\vartheta - r\sin t\sin\vartheta$$

,όπου $t \in \left[k\frac{\pi}{2}, k\pi\right)$ και $k \geq 0$ είναι ένας ακέραιος που καθορίζεται από το λογισμικό έτσι ώστε να σχεδιάζεται πάντα το κατάλληλο μισό του ελλειψοειδούς. Επίσης, θα πρέπει να επισημανθεί το εξής σημαντικό. Επειδή η γλώσσα προγραμματισμού Java υποστηρίζει ένα διαφορετικό

[1] Παρατηρείστε ότι αν $a/2 = r \Rightarrow a = 2r = 160$, τότε το εισερχόμενο τόξο θα είναι ημικύκλιο ακτίνας $r = 80$.

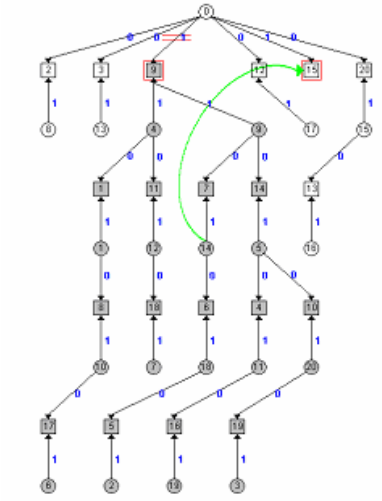
σύστημα συντεταγμένων από αυτό που χρησιμοποιεί η Ευκλείδεια Γεωμετρία[2] **[Kg]**, χρειάστηκε να γίνουν μερικές ασήμαντες αλλαγές στους τύπους που παρουσιάσθηκαν, οι οποίες όμως δε θα αναφερθούν επειδή αφορούν ένα χαρακτηριστικό της συγκεκριμένης γλώσσας προγραμματισμού.



Εικόνα 7.12 – Σχεδιάζοντας την καμπύλη του εισερχόμενου τόξου

Όλοι αυτοί οι μετασχηματισμοί εφαρμόστηκαν με επιτυχία έτσι ώστε να επιτευχθεί μια καλύτερη οπτικοποίηση των αλγορίθμων.

Τέλος, για να δούμε τη λειτουργία του αλγορίθμου σχεδίασης για προβλήματα μεγαλύτερου μεγέθους, παρουσιάζουμε ένα frame που παράγεται από το λογισμικό όταν λύνει ένα πρόβλημα Αντιστοίχησης μεγέθούς 20.



Εικόνα 7.13 – Επιλύνοντας ένα πρόβλημα Αντιστοίχησης μεγέθους 20 με τον αλγόριθμο A3

[2] Στη Java, ένα σημείο *M(x,y)* αυξάνει τις συντεταγμένες του, κινούμενο νοτιοανατολικά και όχι βορειοανατολικά.

Παρατηρείστε ότι υπάρχουν συνολικά 41 κόμβοι στην εικόνα 7.13(20 κόμβοι γραμμές, 20 κόμβοι στήλες και ο ουδέτερος κόμβος 0).

Ακριβώς η ίδια μεθοδολογία απεικόνισης δέντρων στο επίπεδο εφαρμόστηκε και για την οπτικοποίηση των διασχίσεων ενός δέντρου (Inorder, Preorder, Postorder)**[Tr]**. Όλες οι μικροεφαρμογές Java μπορούν να εκτελεσθούν στην παρακάτω διεύθυνση: http://macedonia.uom.gr/~it0067/netpro/index.htm

7.5 ΑΝΑΦΟΡΕΣ

[PP] C. Papamanthou, K. Paparrizos, *"A Visualization of the Primal Simplex Algorithm for the Assignment Problem"*, accepted demo proposal, ITiCSE 2003

[PPS] C. Papamanthou, K. Paparrizos, N. Samaras, *"An Educational Visualization Software for the Assignment Problem"*, submitted for presentation and publication (PCI 2003 – Panhellenic Conference in Informatics)

[BCS] Byrne, D. M., Catrambone, R., Stasko, T. J. "*Do Algorithm Animations Aid Learning?*", 7th Annual Winter Text Conference, Jackson Hole, January, 1996

[Kg] King, K. N. "*Java Programming from the beginning*", New York: W.W. Norton, c2000

[Kl] S. Kalafatoudis, "*Computer Graphics*" Athens 1991, in Greek

[Pk] G. Pekos, "*Applied Mathematics*", Zygos Publications, Thessaloniki 2000, in Greek

[Tr] http://macedonia.uom.gr/~it0067/traversal.htm (*Visualization of Tree Traversals*)

ΚΕΦΑΛΑΙΟ 8

ΣΥΜΠΕΡΑΣΜΑΤΑ

8.1 ΣΧΟΛΙΑΣΜΟΣ ΘΕΩΡΗΤΙΚΩΝ ΑΠΟΤΕΛΕΣΜΑΤΩΝ

Κοιτάζοντας προσεχτικά τη θεωρία των αλγορίθμων που παρουσιάστηκαν στη διπλωματική εργασία, συμπεραίνουμε ότι τα υπολογιστικά αποτελέσματα που εξάχθηκαν επιβεβαιώνουν το θεωρητικό υπόβαθρο. Ο αλγόριθμος Simplex, εμφανίζεται τελικά λιγότερο αποδοτικός από τους αλγορίθμους εξωτερικών σημείων. Αυτό συμβαίνει επειδή ο αλγόριθμος Simplex έχει εκθετική πολυπλοκότητα της τάξης $O(2^n)$ στη χειρότερη περίπτωση. Οι αλγόριθμοι εξωτερικών σημείων που παρουσιάσθηκαν έχουν άνω όριο επαναλήψεων ένα πολυώνυμο του n. Συγκεκριμένα τα άνω όρια επαναλήψεων των αλγορίθμων ενδεικτικά για το πρόβλημα αντιστοίχισης φαίνονται στον παρακάτω πίνακα.

Πίνακας 8.1 – Άνω όριο επαναλήψεων των αλγορίθμων για προβλήματα Αντιστοίχησης μεγέθους n[1]

Αλγόριθμος	Ξεκίνημα	Άνω Όριο Επαναλήψεων
Simplex	Βορειοδυτική Γωνία	$2^n - 1$
Paparrizos	Δέντρο Balinski	$(n-1)(n-2)/2$
Paparrizos	Δάσος AKP	$n(n-1)/2$
Paparrizos	Απλό Ξεκίνημα	$n(n+1)/2$

Βέβαια, ο αναγνώστης θα μπορούσε να επισημάνει ότι δεν υπήρχε λόγος να εκτελεσθεί η υπολογιστική μελέτη, αφού από τη θεωρία φαίνεται ότι οι αλγόριθμοι εξωτερικών σημείων είναι πολύ καλύτεροι και αποδοτικότεροι. Παρόλο αυτά, ο αλγόριθμος simplex εμφανίζει πολυωνυμική συμπεριφορά σε πραγματικά προβλήματα, και γι' αυτό είναι απαραίτητη η εκτέλεση μιας εκτενούς συγκριτικής υπολογιστικής μελέτης.

Τέλος, το ότι οι αλγόριθμοι εξωτερικών σημείων εκτελούν λιγότερες επαναλήψεις από τον αλγόριθμο simplex είναι απολύτως φυσιολογικό αν αναλογισθεί κανείς τον τρόπο που δουλεύουν οι αλγόριθμοι αυτής της κλάσης. Ενώ ο αλγόριθμος simplex κατασκευάζει διαδοχικές βάσεις μεταβαίνοντας από την μια κορυφή του πολυέδρου της εφικτής περιοχής στην άλλη, οι αλγόριθμοι εξωτερικών σημείων αναπτύσσουν δύο δρόμους και χρησιμοποιούν μη εφικτά σημεία. Με βάση αυτή τη γεωμετρική μετακίνηση εξετάζεται μικρότερο πλήθος κορυφών απ' ότι στον αλγόριθμο simplex.

8.2 ΣΧΟΛΙΑΣΜΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΗΣ ΕΜΠΕΙΡΙΑΣ

Όλοι οι αλγόριθμοι προγραμματίστηκαν με αποτελεσματικό τρόπο έτσι ώστε να εξοικονομείται χρόνος και να μην εκτελούνται περιττές εντολές και βρόχοι. Χρησιμοποιήθηκαν ακριβώς οι ίδιες προγραμματιστικές τεχνικές και για τους τέσσερις αλγορίθμους. Ιδιαίτερη βαρύτητα δόθηκε στον τρόπο ανανέωσης των δομών δεδομένων που χρησιμοποιούν οι αλγόριθμοι. Οι δομές δεδομένων σε κάθε επανάληψη δεν υπολογίζονται εξ' αρχής αλλά ανανεώνονται μόνο τα τμήματα των δομών που αλλάζουν. Με αυτόν τον τρόπο ανανεώνονται τα διανύσματα βάθους d και το διάνυσμα πατέρα – κόμβου p. Έτσι, ο χρόνος εκτέλεσης μειώνεται σημαντικά και οι αλγόριθμοι γίνονται ακόμα πιο αποτελεσματικοί.

Παράλληλα, έγινε προσπάθεια να χρησιμοποιηθεί αναδρομικός προγραμματισμός σε περιπτώσεις που χρειαζόταν. Έτσι, η βασικότατη διαδικασία που υπολογίζει το αποκομμένο δέντρο T^*(*Preorder*), υλοποιήθηκε αναδρομικά.

Τέλος, εσκεμμένα δε χρησιμοποιήθηκαν ενσωματωμένες συναρτήσεις και ιδιαίτερα χαρακτηριστικά του Matlab, έτσι ώστε να μπορεί ο αναγνώστης να υλοποιήσει τους αλγορίθμους σε οποιαδήποτε άλλη γλώσσα προγραμματισμού (Pascal, C, C++). Επίσης,

[1] Παρόλο που για n μεγάλο είναι $(n-1)(n-2)/2 < n(n-1)/2 < n(n+1)/2 < 2^n - 1$, η κατάταξη των αλγορίθμων στη μέση περίπτωση δεν ακολουθεί τη διάταξη των άνω ορίων.

χρησιμοποιήθηκε καθαρά δομημένος προγραμματισμός με οργάνωση σε επίπεδο συναρτήσεων και διαδικασιών.

8.3 ΣΧΟΛΙΑΣΜΟΣ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΑΠΟΤΕΛΕΣΜΑΤΩΝ

8.3.1 Πρόβλημα Μεταφοράς

Στο πρόβλημα Μεταφοράς όπως είδαμε τρέξαμε 3 κλάσεις δεδομένων. Η κάθε κλάση αποτελείται από δεδομένα μεγέθους $n \times n$, $n \times 2n$ και $2n \times n$. Μελετώντας τα αποτελέσματα που πήραμε προσεχτικά, παρατηρούμε ότι σε επαναλήψεις η κατάταξη των αλγορίθμων στο πλήθος των περιπτώσεων είναι η εξής (σε αύξουσα διάταξη):

1. T3 (Αλγόριθμος Παπαρρίζου με ξεκίνημα το δάσος AKP)
2. T4 (Αλγόριθμος Παπαρρίζου με απλό ξεκίνημα)
3. T2 (Αλγόριθμος Παπαρρίζου με ξεκίνημα δέντρο Balinski)
4. T1 (Πρωτεύων Αλγόριθμος Simplex)

Σε χρόνο cpu, η κατάταξη είναι η ίδια με τη διαφορά ότι δεν είναι πάντα ξεκάθαρο ποιος αλγόριθμος είναι κάτοχος των θέσεων 3,4. Έτσι, σε κυρίως "hard" δεδομένα και όταν έχουμε μεγέθη προβλημάτων $2n \times n$, ο αλγόριθμος simplex εμφανίζεται αποδοτικότερος σε χρόνο cpu από τον αλγόριθμο T2. Επίσης, σε δεδομένα τέτοιας φύσης, η συμπεριφορά του αλγορίθμου simplex είναι πολύ καλή και σε αριθμό επαναλήψεων και μάλιστα για n μικρά προσεγγίζει (χωρίς να ξεπερνάει) και τους άλλους αλγορίθμους εξωτερικών σημείων.

Επίσης, στην κλάση 3, για δεδομένα $2n \times n$, έχουμε ανατροπή του σκηνικού στο μέσο της εκτέλεσης της υπολογιστικής μελέτης. Αρχικά, ο αλγόριθμος simplex έρχεται δεύτερος σε χρόνο cpu, ενώ για $n \geq 100$ παίρνει την τρίτη θέση, παραχωρώντας τη θέση του στον αλγόριθμο T4.

Ο αλγόριθμος T3 παραμένει σταθερά πρώτος σε όλες τις κλάσεις και είναι μέχρι 4 φορές ταχύτερος από τον αλγόριθμο simplex.

8.3.1 Πρόβλημα Αντιστοίχησης

Το βασικό συμπέρασμα που προκύπτει από τη μελέτη των υπολογιστικών αποτελεσμάτων για το πρόβλημα αντιστοίχησης έρχεται σε πλήρη συμφωνία με αυτό που προέκυψε για το πρόβλημα Μεταφοράς. Σε αριθμό επαναλήψεων, σε όλες τις κλάσεις προβλημάτων η κατάταξη είναι η ακόλουθη (σε αύξουσα διάταξη):

1. A3 (Αλγόριθμος Παπαρρίζου με ξεκίνημα το δάσος AKP)
2. A4 (Αλγόριθμος Παπαρρίζου με απλό ξεκίνημα)
3. A2 (Αλγόριθμος Παπαρρίζου με ξεκίνημα δέντρο Balinski)
5. A1 (Πρωτεύων Αλγόριθμος Simplex)

Σε χρόνο cpu παρατηρείται το εξής. Ενώ για μικρά μεγέθη προβλημάτων ο αλγόριθμος A2 εμφανίζεται πιο αποδοτικός από τον αλγόριθμο simplex, για $n \geq 120$, οι δύο αλγόριθμοι αλλάζουν θέση. Αυτό δε γίνεται μόνο στην 4^{η} κλάση δεδομένων (*Συμμετρικοί πίνακες πραγματικών αριθμών ομοιόμορφα κατανεμημένων στο διάστημα* $[1,100)$*, δηλαδή* $c_{ij} \sim U(1,10^2) \wedge c_{ij} = c_{ji},\ i = 1,...,n,\ j \geq i$), η οποία φαίνεται ότι ωφελεί τους αλγορίθμους εξωτερικών σημείων.

Ο αλγόριθμος A3 παραμένει σταθερά πρώτος και είναι μέχρι 6 φορές ταχύτερος από τον αλγόριθμο simplex.

8.4 ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ

Σε μελλοντικό στάδιο υπάρχει αρκετή ερευνητική εργασία στον επιστημονικό χώρο του Δικτυακού Προγραμματισμού. Παρακάτω αναφέρονται ορισμένα θέματα προς μελλοντική ενασχόληση:

1. Ο προγραμματισμός των αλγορίθμων ώστε να λύνουν και αραιά (*sparse*) προβλήματα, τα οποία αντιπροσωπεύουν ως επί το πλείστο την πραγματικότητα, καθώς και προβλήματα benchmark. Με ανάλογες υπολογιστικές μελέτες, μπορούμε να εξετάσουμε τη συμπεριφορά των αλγορίθμων σε αραιά προβλήματα και να βγάλουμε χρήσιμα συμπεράσματα.
2. Επίσης, ένα άλλο πρόβλημα αποτελεί η γενίκευση των αλγορίθμων εξωτερικών σημείων για άλλα προβλήματα Δικτυακού Προγραμματισμού, όπως το Πρόβλημα Ροής Ελαχίστου Κόστους καθώς και η γενίκευση τους σε άλλα προβλήματα Δικτύων όπως τα Προβλήματα Ελαχίστων Δρόμων.
3. Η εύρεση ενός τρόπου υπολογισμού προβλημάτων Δικτυακού Προγραμματισμού τα οποία θα επιλύονται στο μέγιστο αριθμό επαναλήψεων εφαρμόζοντας τους αλγορίθμους εξωτερικών σημείων.
4. Ο παράλληλος προγραμματισμός των αλγορίθμων (π.χ. με MPI ή grid), χρησιμοποιώντας διαφορετικές τοπολογίες επεξεργαστών.
5. Ο υπολογισμός της πολυπλοκότητας μέσης περίπτωσης για τους αλγορίθμους εξωτερικών σημείων.