ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΤΟΜΕΑΣ ΟΡΓΑΝΩΣΗΣ, ΠΑΡΑΓΩΓΗΣ & ΒΙΟΜΗΧΑΝΙΚΗΣ ΔΙΟΙΚΗΣΗΣ

Ένας αλγόριθμος branch and bound για μια ειδική περίπτωση του ακέραιου προβλήματος σακιδίου με περιορισμούς πολλαπλών επιλογών

Διπλωματική εργασία

υπό

ΑΝΝΑΣ – ΕΙΡΗΝΗΣ ΠΑΠΑΓΕΩΡΓΙΟΥ

ΚΩΝΣΤΑΝΤΙΝΟΥ ΒΕΡΙΛΛΗ

Υπεβλήθη για την εκπλήρωση μέρους των απαιτήσεων για την απόκτηση του Διπλώματος Μηχανολόγου Μηχανικού

Βόλος, Μάρτιος 2016

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΒΙΒΛΙΟΘΗΚΗ & ΚΕΝΤΡΟ ΠΛΗΡΟΦΟΡΗΣΗΣ
ΕΙΔΙΚΗ ΣΥΛΛΟΓΗ «ΓΚΡΙΖΑ ΒΙΒΛΙΟΓΡΑΦΙΑ»

Αριθ. Εισ.:	14730/1
Ημερ. Εισ.:	22-03-2017
Δωρεά:	Συγγραφέας
Ταξιθετικός Κωδικός:	ΠΤ – ΜΜ
	2016
	ΠΑΠ

Copyright© 2016 Παπαγεωργίου Άννα – Ειρήνη, Βερίλλης Κωνσταντίνος

Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τους συγγραφείς.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανολόγων Μηχανικών της Πολυτεχνικής Σχολής του Πανεπιστημίου Θεσσαλίας δεν υποδηλώνει αποδοχή των απόψεων των συγγραφέων.

Έγκριση

Εγκρίθηκε από τα Μέλη της Τριμελούς Εξεταστικής Επιτροπής:

Πρώτος Εξεταστής (Επιβλέπων)
Δρ. Γεώργιος Κοζανίδης
Επίκουρος Καθηγητής
Τμήμα Μηχανολόγων Μηχανικών
Πανεπιστήμιο Θεσσαλίας

Δεύτερος Εξεταστής
Δρ. Σπυρίδων Καραμάνος
Καθηγητής
Τμήμα Μηχανολόγων Μηχανικών
Πανεπιστήμιο Θεσσαλίας

Τρίτος Εξεταστής
Δρ. Δημήτριος Παντελής
Επίκουρος Καθηγητής
Τμήμα Μηχανολόγων Μηχανικών
Πανεπιστήμιο Θεσσαλίας

Ευχαριστίες

Πρώτα από όλα, θα θέλαμε να εκφράσουμε τις ευχαριστίες μας στον κ. Γιώργο Κοζανίδη, Επίκουρο Καθηγητή του Πανεπιστημίου Θεσσαλίας, για την εμπιστοσύνη που μας έδειξε και την ευκαιρία που μας έδωσε να ασχοληθούμε με το αντικείμενο της επιχειρησιακής έρευνας καθώς και για την καθοδήγηση του σε όλα τα στάδια της εκπόνησης της διπλωματική εργασίας μας.

Επίσης, ευχαριστούμε και τα υπόλοιπα δύο μέλη της εξεταστικής επιτροπής της διπλωματικής εργασίας μας, κ. Σπύρο Καραμάνο, Καθηγητή του Τμήματος Μηχανολόγων Μηχανικών, και κ. Δημήτρη Παντελή, Επίκουρο Καθηγητή του ίδιου τμήματος, για τον χρόνο που αφιέρωσαν για την ανάγνωση της εργασίας μας.

Επίσης, θα θέλαμε να ευχαριστήσουμε και να εκφράσουμε την ευγνωμοσύνη μας στις οικογένειες μας για τη στήριξη, υλική και ηθική, που μας παρείχαν καθ' όλη τη διάρκεια των σπουδών μας.

Τέλος, θα θέλαμε να ευχαριστήσουμε τους φίλους μας, ιδιαίτερα τον Γιώργο και τη Χριστίνα, που χωρίς αυτούς θα ήταν δύσκολη η επίτευξη του στόχου μας.

Άννα – Ειρήνη Παπαγεωργίου

Κωνσταντίνος Βερίλλης

Ένας αλγόριθμος branch and bound για μια ειδική περίπτωση του ακέραιου προβλήματος σακιδίου με περιορισμούς πολλαπλών επιλογών

ΑΝΝΑ – ΕΙΡΗΝΗ ΠΑΠΑΓΕΩΡΓΙΟΥ

ΚΩΝΣΤΑΝΤΙΝΟΣ ΒΕΡΙΛΛΗΣ

Πανεπιστήμιο Θεσσαλίας, Τμήμα Μηχανολόγων Μηχανικών

Επιβλέπων καθηγητής: Δρ. Γεώργιος Κοζανίδης

Επίκουρος Καθηγητής, Πανεπιστήμιο Θεσσαλίας, Τμήμα Μηχανολόγων Μηχανικών

Περίληψη

Στην παρούσα εργασία μελετάται ένα ειδικό πρόβλημα λήψης αποφάσεων το οποίο παρουσιάζεται στην περιοχή της διαχείρισης αποθεμάτων για βέλτιστη τιμολόγηση προϊόντων με σκοπό την εκποίησή τους εντός ενός συγκεκριμένου χρονικού ορίζοντα. Το πρόβλημα μορφοποιείται ως ένα μοντέλο βελτιστοποίησης που παίρνει τη μορφή ενός ακέραιου προβλήματος σακιδίου με περιορισμούς πολλαπλών επιλογών (integer multiple choice knapsack problem). Με βάση τις ειδικές ιδιότητες αυτού του προβλήματος, αναπτύσσονται σημαντικά θεωρητικά αποτελέσματα για το μοντέλο αυτό, και χρησιμοποιούνται για την ανάπτυξη ενός εξειδικευμένου αλγόριθμου branch and bound για την εύρεση της βέλτιστης λύσης. Ο αλγόριθμος εκμεταλλεύεται την ειδική σχέση που υπάρχει ανάμεσα σε γειτονικούς κόμβους του δέντρου αναζήτησης, η οποία καθιστά εφικτή την επίλυση των αντίστοιχων γραμμικών χαλαρώσεων με μεγάλη αποτελεσματικότητα.

Λέξεις κλειδιά: τιμολόγηση προϊόντων, ακέραιο πρόβλημα σακιδίου, περιορισμοί πολλαπλών επιλογών, αλγόριθμος branch and bound.

A branch and bound algorithm for a specific case of an integer knapsack problem with multiple choice constraints

ANNA-EIRINI PAPAGEORGIOU

KONSTANTINOS VERILLIS

University of Thessaly, Department of Mechanical Engineering

Supervisor: Dr. George Kozanidis

Assistant Professor, Department of Mechanical Engineering, University of Thessaly

Abstract

An interesting decision making problem that arises in inventory management for optimal pricing of an inventory of products which must be sold over a pre-determined phase-out period is addressed. The problem is formulated as an optimization model which takes the form of an integer multiple choice knapsack problem. Based on the special properties of the problem, several important theoretical results are proven and utilized to develop an exact branch and bound algorithm for finding the optimal solution. The algorithm exhibits high efficiency due to the fact that it exploits the special relationship between parent and children nodes of the branch and bound tree in order to find the optimal solution to the LP relaxation of the associated sub-problems.

Keywords: product pricing; integer knapsack problem; multiple choice constraints; branch-and-bound algorithm.

Περιεχόμενα

Περιεχόμενα πινάκων

Περιεχόμενα διαγραμμάτων

Περιεχόμενα σχημάτων

1. Εισαγωγή

1.1 Σκοπός εργασίας

Ένα από τα πλέον δύσκολα προβλήματα στην περιοχή της επιχειρησιακής έρευνας είναι και το πρόβλημα του σακιδίου (knapsack problem - KP), στο οποίο πρέπει να γίνει επιλογή ενός υποσυνόλου αντικειμένων που θα εισαχθούν σε ένα σακίδιο έτσι ώστε να μεγιστοποιηθεί η συνολική τους αξία χωρίς να παραβιαστεί η χωρητικότητα του σακιδίου. Το πρόβλημα μορφοποιείται ως ένα μαθηματικό μοντέλο ακέραιου προγραμματισμού, ενώ για την επίλυσή του έχει αναπτυχθεί πληθώρα εξειδικευμένων αλγορίθμων βελτιστοποίησης.

Το πρόβλημα του σακιδίου και οι πάμπολλες παραλλαγές του που έχουν αναπτυχθεί απαντώνται σε πολλές εφαρμογές της επιχειρησιακής έρευνας, όπως φόρτωση εμπορευμάτων, προγραμματισμός παραγωγής, κατανομή πόρων, κτλ. (Rouf, 2005). Έστω ότι υπάρχουν n αντικείμενα όπου η αξία του αντικειμένου i είναι v_i και ο όγκος του w_i. Αν ο μέγιστος όγκος των αντικειμένων που θα μπουν στο σακίδιο είναι b, ζητούνται οι τιμές των μεταβλητών x_i που μεγιστοποιούν την αντικειμενική $\sum_{i=1}^{n} x_i\, v_i$ υπό τον περιορισμό $\sum_{i=1}^{n} x_i w_i \leq b$ με $x_i \in \{0,1\}$. Πέραν των δυαδικών, το KP μπορεί να μορφοποιηθεί και να λυθεί και για συνεχείς και για γενικές ακέραιες μεταβλητές.

Το πρόβλημα σακιδίου με περιορισμούς πολλαπλών επιλογών (multiple choice knapsack problem – MCKP) είναι μία από τις πιο εκτενώς ερευνημένες παραλλαγές του προβλήματος του σακιδίου τις τελευταίες δεκαετίες (Lin, 1998a). Σε σχέση με το κλασσικό πρόβλημα σακιδίου, το MCKP περιλαμβάνει επιπλέον περιορισμούς οι οποίοι χωρίζουν τα αντικείμενα σε διακριτές ομάδες και επιβάλλουν ότι μόνο ένα αντικείμενο μπορεί να επιλεγεί από κάθε ομάδα.

Προβλήματα κατανομής πόρων χωρίς την υπέρβαση ενός ποσού με σκοπό τη μεγιστοποίηση των συνολικών κερδών καθώς και υποπεριπτώσεις τους που βασίζονται στο πρόβλημα του σακιδίου με πολλαπλούς περιορισμούς και περιορισμούς ακεραιότητας έχουν αποτελέσει αντικείμενο έρευνας πολλών ετών. Πρόκειται για NP-hard προβλήματα που μπορούν να λυθούν με πολλές μεθόδους οι πιο συνηθισμένες από τις οποίες περιλαμβάνουν τεχνικές δυναμικού προγραμματισμού, branch and bound, branch and cut ή branch and price.

Οι branch and bound αλγόριθμοι λύνουν τα προβλήματα μέσω ενός δέντρου αναζήτησης. Αρχικά στον πρώτο κόμβο του δέντρου λύνεται η γραμμική χαλάρωση του προβλήματος

αγνοώντας τους περιορισμούς ακεραιότητας. Η λύση αυτή αποτελεί και ένα άνω φράγμα (για πρόβλημα μεγιστοποίησης) στη βέλτιστη τιμή της αντικειμενικής. Στη συνέχεια επιλέγεται μία μη ακέραια μεταβλητή στην οποία επιβάλλονται περιορισμοί ακεραιότητας και έτσι προκύπτουν δύο διακλαδώσεις, μία για το άνω και μία για το κάτω όριο. Συνεχίζεται έτσι η διαδικασία και κάθε φορά επιλέγεται ένας κόμβος για διακλάδωση και λύνεται η γραμμική χαλάρωση του αντίστοιχου προβλήματος. Αν σε ένα κόμβο βρεθεί ακέραια λύση τότε η τιμή της αντικειμενικής αποτελεί και ένα κάτω φράγμα για τη βέλτιστη αντικειμενική του προβλήματος και τότε ο κόμβος αυτό δεν αναπτύσσεται περαιτέρω (Κοζανίδης, 2008). Ομοίως και σε περίπτωση που δεν υπάρχει εφικτή λύση στη γραμμική χαλάρωση. Συνήθως επιλέγεται για διακλάδωση ο κόμβος με το καλύτερο όριο στη βέλτιστη τιμή της αντικειμενικής, αλλά υπάρχουν και άλλες στρατηγικές για την επιλογή αυτή. Η διαδικασία τερματίζεται όταν δεν υπάρχουν άλλοι κόμβοι προς εξερεύνηση.

Στην παρούσα εργασία μελετήθηκε ένα πρόβλημα διαχείρισης αποθεμάτων. Συγκεκριμένα, αναζητήθηκαν οι βέλτιστοι χρόνοι εφαρμογής τιμών για ένα απόθεμα προϊόντων με σκοπό την εκποίησή τους εντός ενός συγκεκριμένου χρονικού ορίζοντα. Τα ερωτήματα είναι πότε και για πόσο καιρό θα εφαρμοστούν οι προκαθορισμένες αυτές τιμές ώστε να μεγιστοποιηθούν τα κέρδη και ταυτόχρονα ο συνολικός όγκος πωλήσεων κατά τη διάρκεια των περιόδων αυτών να μην ξεπεράσει το συνολικό απόθεμα. Εκτός από την περίοδο εκποίησης, αναζητείται και το εναπομένον απόθεμα στο τέλος του ορίζοντα, μιας και υπάρχει η δυνατότητα να πουληθεί σε μία συγκεκριμένη τιμή σε έναν ανεξάρτητο έμπορο (Zaarour, 2011).

Για την επίλυση του προβλήματος αυτού, η σχέση τιμής - ζήτησης δεν έπαιξε ρόλο καθώς θεωρήθηκαν γνωστές οι τιμές για κάθε χρονική περίοδο της διαδικασίας εκποίησης καθώς και τα αντίστοιχα έσοδα και ο όγκος πωλήσεων που προκύπτουν από την εφαρμογή τους. Ήταν δεδομένη επίσης η τιμή πώλησης του αποθέματος μετά την περίοδο εκποίησης, το συνολικό αρχικό απόθεμα και ο διαθέσιμος χρόνος για την εφαρμογή τιμών.

Το πρόβλημα μορφοποιείται ως ένα μοντέλο βελτιστοποίησης που παίρνει τη μορφή ενός ακέραιου προβλήματος σακιδίου με περιορισμούς πολλαπλών επιλογών και για την επίλυσή του αναπτύχθηκε ένας αλγόριθμος branch and bound ο οποίος κάνει χρήση των ιδιοτήτων των μεταβλητών και του προβλήματος για την γρήγορη επίλυση των κόμβων του δέντρου αναζήτησης. Μία προηγούμενη έκδοση της εργασίας παρουσιάστηκε σε ένα επιστημονικό συνέδριο (Παπαγεωργίου, Βερίλλης, & Κοζανίδης, 2015).

1.2 Οργάνωση εργασίας

Η εργασία αυτή αποτελείται από 7 ενότητες οι οποίες περιλαμβάνουν:

- Την εισαγωγή που περιέχει:
  - τον σκοπό της εργασίας
  - τη δομή της παρούσας εργασίας
- Την βιβλιογραφική ανασκόπηση.
- Την μαθηματική μορφοποίηση του ακεραίου προβλήματος και οι επεξηγήσεις για τις παραμέτρους.
- Την μεθοδολογία επίλυσης, στην οποία βρίσκονται:
  - η λεπτομερής περιγραφή της αρχικής μεθοδολογίας επίλυσης,
  - οι αλλαγές που βελτιώνουν την μεθοδολογία,
  - ένα αριθμητικό παράδειγμα για την κατανόηση του αλγορίθμου.
- Την περιγραφή του κώδικα που δίνει μία σύντομη περιγραφή του κώδικα που υλοποιήθηκε για την μεθοδολογία.
- Τα υπολογιστικά πειράματα που εκπονήθηκαν με χρήση του κώδικα.
- Τα συμπεράσματα και τις προτάσεις για μελλοντική έρευνα.

Τέλος, στο παράρτημα δίνεται ο κώδικας του οποίου η περιγραφή έγινε παραπάνω.

2. Βιβλιογραφική ανασκόπηση

Το πρόβλημα σακιδίου και οι περιπτώσεις του έχουν ερευνηθεί εκτενώς τις τελευταίες δεκαετίες και υπάρχει για αυτό πλούσια βιβλιογραφία. Ο Nauss πρότεινε δύο αλγορίθμους branch and bound που χρησιμοποιούσαν διαφορετικούς τρόπους για να βρεθούν χαλαρώσεις (Nauss, 1978), ενώ οι Sinha και Zoltners πρότειναν έναν αλγόριθμο που ξεκινά με την απαλοιφή μεταβλητών οι οποίες έχουν οπωσδήποτε μηδενική τιμή στη βέλτιστη λύση (Sinha & Zoltners, 1979). Με τα αποτελέσματα προβλημάτων μεγάλης κλίμακας ασχολήθηκαν οι Armstrong et al. εφαρμόζοντας μία διαδικασία παρόμοια με εκείνη των Sinha και Zoltners (Armstrong, Kung, Sinha, & Zoltners, 1983).

Εκτός από έρευνες που αφορούν το κλασσικό πρόβλημα σακιδίου, έχουν αναπτυχθεί πολλές και για τις παραλλαγές του και ο Lin έχει συγκεντρώσει και αξιολογήσει κάποιες από αυτές (Lin, 1998b). Οι Dyer et al. ανέπτυξαν έναν branch and bound αλγόριθμο βασισμένο σε μία μέθοδο για τη λύση του γραμμικού KP με περιορισμούς πολλαπλών επιλογών με αναζήτηση τύπου depth-first (Dyer, Kayal, & Walker, 1984). Έτσι, μπορούσαν να ελαττώσουν το μέγεθος του προβλήματος αφότου βρισκόταν μία αρχική εφικτή δυαδική λύση. Με τη χαλάρωση Lagrange ασχολήθηκαν και οι Aggarwal et al. οι οποίοι πρότειναν έναν αλγόριθμο δύο σταδίων, όπου αφότου καθοριστεί η βέλτιστη τιμή του πολλαπλασιαστή Lagrange, κατατάσσονται οι λύσεις μέσω ενός branch and bound αλγορίθμου (Aggarwal, Deo, & Sarkar, 1992).

Ο Pisinger παρουσίασε έναν αλγόριθμο που βασίζεται στη θεωρία του ελάχιστου πυρήνα (minimal core) (Pisinger, 1995), ενώ οι Dyer et al. ανέπτυξαν έναν υβριδικό αλγόριθμο που συνδυάζει δυναμικό προγραμματισμό και branch and bound για να εφαρμόσει τα όρια σε κάθε κόμβο του δέντρου και να μειώσει έτσι το μέγεθος του προβλήματος (Dyer, Riha, & Walker, 1995).

Με την επίλυση ενός γραμμικού προβλήματος σακιδίου με περιορισμούς άνω ορίου ασχολήθηκαν και οι Glover et al. οι οποίοι ανέπτυξαν μία μέθοδο που βασίζεται στη δυϊκή simplex (Glover, & Klingman, 1979). Βασισμένος στην ίδια μέθοδο είναι και ο αλγόριθμος που ανέπτυξαν οι Sarin et al. για την επίλυση προβλημάτων σακιδίου με πολλαπλούς περιορισμούς. (Sarin, & Karwan, 1989).

3. Μαθηματική μορφοποίηση

Ο αλγόριθμος που θα παρουσιαστεί εφαρμόζεται σε ένα ακέραιο πρόβλημα σακιδίου πολλαπλών επιλογών με την εξής μορφοποίηση:

$$Max \sum_{i=1}^{n} R_i \boldsymbol{x_i} + C\boldsymbol{r} \qquad (1)$$

$$s.t \quad \sum_{i=1}^{n} V_i \boldsymbol{x_i} + \boldsymbol{r} \leq I \qquad (2)$$

$$\sum_{i=1}^{n} \boldsymbol{x_i} \leq T \qquad (3)$$

$$\boldsymbol{r},\ \boldsymbol{x_i}\ (i = 1, \dots, n) \in Z^{+} \qquad (4)$$

Για το πρόβλημα αυτό αναζητούνται οι βέλτιστοι χρόνοι εφαρμογής συγκεκριμένων τιμών εκποίησης καθώς και το απόθεμα στο τέλος του παραπάνω χρονικού ορίζοντα με σκοπό τη μεγιστοποίηση των εσόδων από τη διαδικασία εκποίησης.

Οι μεταβλητές απόφασης x_i, για $i = 1,2, \dots n$, αντιστοιχούν στον αριθμό των χρονικών περιόδων στις οποίες θα εφαρμοστεί η τιμή πώλησης P_i, ενώ η r στο διαθέσιμο απόθεμα μετά το πέρας της περιόδου εκποίησης.

Οι τιμές P_i ακολουθούν φθίνουσα πορεία καθ' όλη τη διαδικασία εκποίησης με $P_1 > P_2 > \cdots > P_n > C$, όπου C είναι η τιμή πώλησης στο τέλος του χρονικού ορίζοντα. R_i είναι τα έσοδα που προκύπτουν από την εφαρμογή της τιμής πώλησης P_i σε μία χρονική περίοδο. Οι παράμετροι R_i βρίσκονται από τη σχέση $R_i = \alpha P_i^{\beta+1}$ όπου α είναι μία σταθερά και β η ελαστικότητα της ζήτησης, με $\alpha > 0$ και $\beta < -1$. Λόγω αυτού τα έσοδα αυξάνονται κατά τη διάρκεια της διαδικασίας εκποίησης και είναι $R_1 < R_2 < \cdots < R_n$.

Ο πρώτος περιορισμός του προβλήματος (2) σκοπό έχει να εξασφαλίσει ότι ο συνολικός όγκος πωλήσεων σε όλη τη περίοδο εκποίησης δεν θα υπερβεί το διαθέσιμο απόθεμα I, με V_i τον όγκο πωλήσεων που αντιστοιχεί σε κάθε χρονική περίοδο. Οι παράμετροι V_i σχετίζονται με την τιμή με την παρακάτω σχέση $V_i = \alpha P_i^{\beta}$ και, όπως και τα έσοδα, αυξάνονται: $V_1 < V_2 < \cdots < V_n$ (Zaarour, 2011). Ο δεύτερος περιορισμός του προβλήματος (3) εξασφαλίζει ότι ο συνολικός αριθμός περιόδων εφαρμογής των διάφορων τιμών

πώλησης δεν θα υπερβεί το μέγεθος του χρονικού ορίζοντα T. Τέλος, ο περιορισμός (4) επιβάλει την ακεραιότητα στις μεταβλητές απόφασης.

4. Μεθοδολογία επίλυσης

4.1 Αρχική μεθοδολογία επίλυσης

Για την επίλυση του ακέραιου αυτού προβλήματος έχει αναπτυχθεί ένας αλγόριθμος βασισμένος στον αλγόριθμο που αναπτύχθηκε από τους Kozanidis and Melachrinoudis (2004) για το δυαδικό πρόβλημα σακιδίου με γραμμικούς περιορισμούς πολλαπλών επιλογών. Αρχικά, ο αλγόριθμος ταξινομεί τις μεταβλητές απόφασης σε μία λίστα με βάση το λόγο του κέρδους προς τον αντίστοιχο όγκο πωλήσεων. Στη συνέχεια, βρίσκει τη λύση στη γραμμική χαλάρωση του προβλήματος και τέλος, μέσω ενός δέντρου branch and bound, βρίσκει τη βέλτιστη ακέραια λύση.

4.1.1 Δημιουργία λίστας

Για τις ανάγκες του αλγορίθμου θα δημιουργηθεί μία λίστα που θα περιλαμβάνει όλες τις μεταβλητές απόφασης με κάποια συγκεκριμένη σειρά. Η λίστα αυτή θα βοηθήσει μετέπειτα στη γρηγορότερη επίλυση των υπο-προβλημάτων του δέντρου αναζήτησης.

Για όλες τις μεταβλητές απόφασης θα οριστεί ένας λόγος σύμφωνα με τον οποίο θα μπαίνουν στη λίστα. Πρώτα θα μπουν σε σωστή σειρά οι x_i. Για να μπει η πρώτη υπολογίζεται για καθεμία ένας ανεξάρτητος λόγος που είναι ίσος με το συντελεστή της στην αντικειμενική προς το συντελεστή της στον πρώτο περιορισμό $\frac{R_i}{V_i}$ ή αλλιώς από το λόγο των εσόδων που θα προκύψουν από την εφαρμογή της τιμής πώλησης P_i προς τον όγκο πωλήσεων για αυτή τη χρονική περίοδο. Η μεταβλητή με τον μεγαλύτερο ανεξάρτητο λόγο είναι και αυτή που θα εισαχθεί πρώτη στη λίστα.

Για να βρεθεί η δεύτερη μεταβλητή που θα εισαχθεί, θα βρεθεί για κάθε x_i ο λόγος της σε σχέση με τη τελευταία μεταβλητή $x_j, (j \neq i)$ που μπήκε στη λίστα. Ο λόγος αυτός ορίζεται ως εξής: $\frac{R_i - R_j}{V_i - V_j}$. Η μεταβλητή με το μεγαλύτερο λόγο είναι αυτή που μπαίνει δεύτερη στη λίστα. Ομοίως συνεχίζεται η διαδικασία μέχρι να μπουν σε σωστή σειρά όλες οι x_i με το μόνο που αλλάζει να είναι η x_j - η τελευταία δηλαδή μεταβλητή που έχει μπει κάθε φορά στη λίστα.

Η τελευταία κίνηση για να ολοκληρωθεί η λίστα είναι να εισαχθεί στη σωστή θέση η μεταβλητή r σύμφωνα με τη τιμή του λόγου της, ο οποίος είναι C, μια και ο συντελεστής της στον πρώτο περιορισμό είναι 1.

4.1.2 Διαδικασία επίλυσης γραμμικής χαλάρωσης προβλήματος

Μετά τη δημιουργία της λίστας, σειρά έχει η εύρεση της γραμμικής χαλάρωσης του προβλήματος. Για να βρεθεί η λύση θα χρησιμοποιούνται οι όροι I_{res} και T_{res} που περιγράφουν το υπόλοιπο του δεξιού μέλους του πρώτου και δεύτερου περιορισμού αντίστοιχα. Από τους δύο αυτούς όρους θα απασχολήσει περισσότερο το υπόλοιπο του πρώτου περιορισμού I_{res} διότι ο αλγόριθμος τελειώνει όταν μηδενιστεί. Έτσι το $I_{res} = 0$ θα είναι η συνθήκη που θα ελέγχεται μετά από κάθε κίνηση για να αποφασιστεί αν θα συνεχιστεί ο αλγόριθμος ή αν έχει βρεθεί η γραμμική χαλάρωση στο πρόβλημα.

Το πρώτο βήμα για την εύρεση της λύσης είναι η αρχικοποίηση του I_{res}. Από τη στιγμή που σε καμιά μεταβλητή δεν έχει ανατεθεί τιμή, το διαθέσιμο απόθεμα είναι το αρχικό. Δηλαδή $I_{res} = I$.

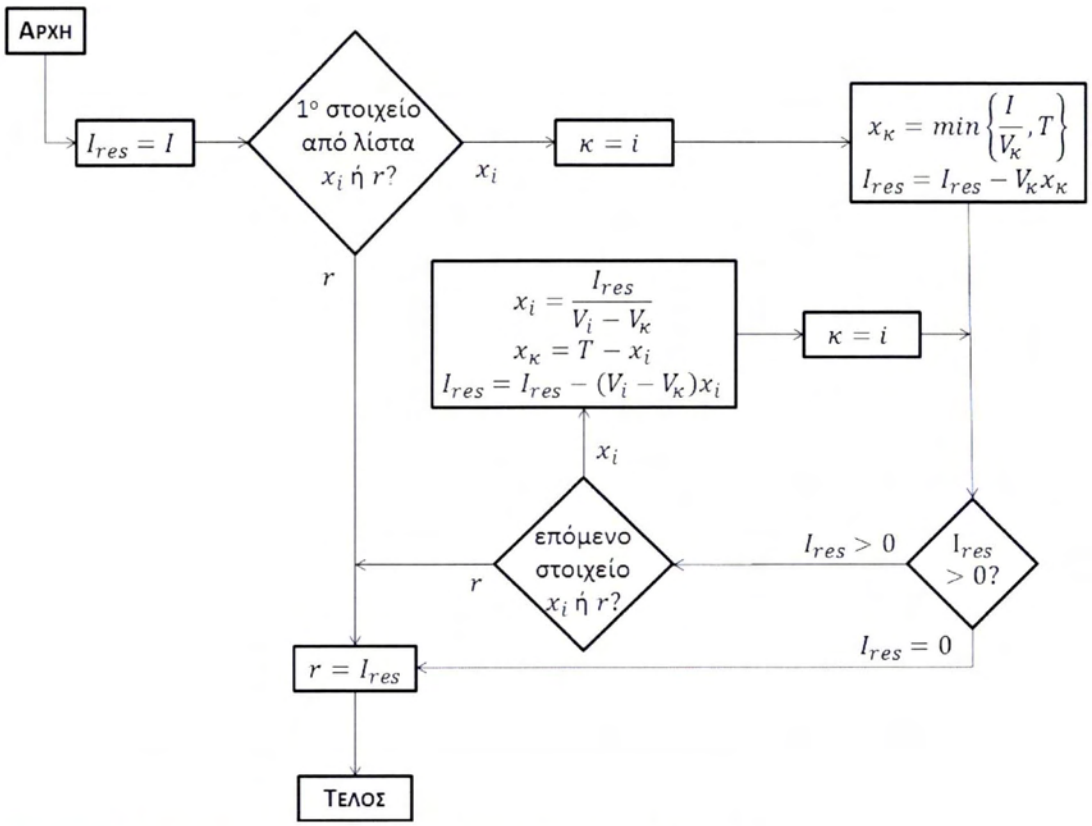
Στη συνέχεια, η σειρά που έχουν οι μεταβλητές απόφασης στη λίστα είναι αυτή που θα υποδείξει ποια θα είναι η πρώτη μεταβλητή που θα μπει στη λύση. Ξεκινώντας από την πρώτη μεταβλητή θα πρέπει να γίνει ο διαχωρισμός για το αν είναι r ή x_i. Αν πρόκειται για την r, τότε μπορεί να ανατεθεί σε αυτήν όλο το I_{res}, μια και δεν υπάρχει κανένας περιορισμός για τη συγκεκριμένη μεταβλητή σε αυτή τη φάση. Από την άλλη, αν η μεταβλητή είναι x_i δεν γίνεται να αυξηθεί απεριόριστα αφού συμμετέχει και στο δεύτερο περιορισμό. Έστω λοιπόν x_k η πρώτη μεταβλητή στη λίστα. Η τιμή που θα πάρει θα είναι η μικρότερη των εξής δύο επιλογών: του υπολοίπου του πρώτου περιορισμού – που για την πρώτη επανάληψη είναι I - προς το συντελεστή της μεταβλητής στον ίδιο περιορισμό $\frac{I}{V_k}$ και του δεξιού μέλους του δεύτερου περιορισμού που είναι T. Η ανάθεση τιμής στην πρώτη μεταβλητή έχει ως αποτέλεσμα την αλλαγή των δεξιών μελών των περιορισμών, συγκεκριμένα το I_{res} θα μειωθεί κατά $V_k x_k$. Θα είναι δηλαδή το καινούριο υπόλοιπο $I_{res} = I_{res} - V_k x_k$.

Έχοντας ολοκληρώσει την πρώτη επανάληψη πρέπει να ελεγχθεί το I_{res}. Αν είναι μηδέν, όπως έχει αναφερθεί προηγουμένως, έχει βρεθεί η λύση. Αν όμως υπάρχει ακόμα διαθέσιμο απόθεμα, πρέπει να συνεχιστούν οι επαναλήψεις ώστε αυτό να ανατεθεί στην επόμενη μεταβλητή στη λίστα.

Αν η επόμενη μεταβλητή είναι r, τότε μπορεί να ανατεθεί σε αυτή όλο το υπόλοιπο I_{res} και να τελειώσει η διαδικασία εύρεσης της γραμμικής χαλάρωσης. Αντίθετα αν είναι x_i τότε η αύξησή της θα πρέπει να γίνει με ταυτόχρονη μείωση της x_k που έχει μπει στη λύση στην προηγούμενη επανάληψη. Η ταυτόχρονη αυτή μεταβολή θα εξασφαλίσει ότι δεν θα

αλλάξει το T_{res} και έτσι το άθροισμα των μεταβλητών θα παραμείνει ίδιο και ίσο με T. Έτσι, η τιμή που θα πάρει θα εξαρτάται από το υπόλοιπο του πρώτου περιορισμού και από τους συντελεστές των x_k και x_i στον ίδιο περιορισμό, και θα είναι $x_i = \frac{I_{res}}{V_i - V_\kappa}$. Η νέα τιμή της x_k πλέον θα είναι $x_\kappa = T - x_i$ και τέλος το I_{res} θα αλλάξει και θα γίνει $I_{res} = I_{res} - (V_i - V_\kappa)x_i$.

Στη συνέχεια θα ελεγχθεί ξανά το I_{res}. Οι ίδιες κινήσεις θα πρέπει να ακολουθηθούν αν είναι διάφορο του μηδενός. Η μόνη διαφορά είναι ότι κάθε φορά που η επόμενη μεταβλητή προς αύξηση στη λίστα είναι x_i, οι παραπάνω υπολογισμοί θα γίνονται με την x_k να είναι η τελευταία που μπήκε στη λύση. Ο αλγόριθμος θα συνεχιστεί μέχρι να μηδενιστεί το I_{res}, όπως φαίνεται στο διάγραμμα 1 παρακάτω.



Διάγραμμα 1 **Διάγραμμα ροής για την επίλυση της γραμμικής χαλάρωσης.**

4.1.3 Branch and bound

Μετά την εύρεση της γραμμικής χαλάρωσης του προβλήματος σειρά έχει η κατάστρωση του δέντρου branch and bound. Επιλέγεται αρχικά μία μεταβλητή για διακλάδωση. Σε αυτό το στάδιο θα χρησιμοποιηθεί η λίστα και οι ιδιότητές της για τη γρηγορότερη εύρεση των

γραμμικών χαλαρώσεων των κόμβων. Συγκεκριμένα, η λύση των δύο υπο-προβλημάτων που προκύπτουν είναι όμοια με τη λύση του κόμβου γονέα με εξαίρεση τη τελευταία αλληλουχία επαναλήψεων.

Για την επίλυση ενός κόμβου, σε πρώτη φάση, αναιρείται η τελευταία επανάληψη του κόμβου γονέα ώστε να ικανοποιηθούν τα όρια που έχουν μόλις εφαρμοστεί σε μία μεταβλητή. Αν πρόκειται για άνω όριο τότε η τιμή της αντίστοιχης μεταβλητής πρέπει να μειωθεί ενώ αν πρόκειται για κάτω όριο, η τιμή θα αυξηθεί. Σε κάθε περίπτωση αυτή η μεταβολή θα επιφέρει αλλαγές στους περιορισμούς και πλέον ένα ή και τα δύο δεξιά μέλη δεν είναι μηδέν. Ανάλογα με τις τιμές που θα πάρουν θα πρέπει να γίνουν οι ακόλουθες κινήσεις όπως φαίνονται παρακάτω.

Αρνητικό I_{res} και T_{res}

Στην περίπτωση που και τα δύο δεξιά μέλη των περιορισμών I_{res} και T_{res} είναι αρνητικά, θα πρέπει να διορθωθεί πρώτα ο δεύτερος περιορισμός. Συγκεκριμένα, θα πρέπει να βρεθεί η μεταβλητή x_i (η r δεν συμμετέχει στο δεύτερο περιορισμό) που έχει θετική τιμή, μεγαλύτερη του κάτω ορίου της, και της οποίας ο ανεξάρτητος λόγος $\frac{R_i}{V_i}$ είναι μικρότερος των υπολοίπων. Για να ικανοποιηθεί ο περιορισμός θα μειωθεί η μεταβλητή αυτή κατά T_{res} ή όσο της επιτρέπει το κάτω όριό της. Αν μετά τις αλλαγές το T_{res} είναι ακόμα αρνητικό, τότε θα αναζητηθεί εκ νέου μία μεταβλητή x_i για μείωση. Οι μεταβολές στις x_i γίνονται μέχρι να επιτευχθεί μηδενικό T_{res}. Η διαδικασία αυτή θα μεταβάλει όμως και την τιμή του I_{res}, με αποτέλεσμα να απαιτείται η διόρθωση του πρώτου περιορισμού όπως φαίνεται παρακάτω.

Αρνητικό I_{res}

Αν το I_{res} είναι αρνητικό, έχει παραβιαστεί ο πρώτος περιορισμός και πρέπει να μειωθεί μία μεταβλητή γιατί έχει γίνει υπέρβαση του διαθέσιμου αποθέματος. Όπως και παραπάνω, πρέπει να συγκριθούν οι λόγοι – αναζητώντας τον μικρότερο, ώστε να βρεθεί η αντίστοιχη μεταβλητή που θα μειωθεί. Για τη μεταβλητή απόφασης r, ο λόγος είναι C. Για τις μεταβλητές απόφασης x_i, αναζητείται μία με θετική τιμή για να μειωθεί και μία αριστερά της στη λίστα για να αυξηθεί, προσέχοντας να μην παραβιαστούν τα άνω ή κάτω όρια κάθε μιας. Η ταυτόχρονη αύξηση/μείωση γίνεται για να διατηρηθεί το δεξί μέλος του δεύτερου περιορισμού T_{res}. Για κάθε συνδυασμό μεταβλητών θα βρίσκεται ο λόγος $\frac{R_k - R_j}{V_k - V_j}$

όπου με k συμβολίζεται η μεταβλητή που βρίσκεται πιο δεξιά – και άρα θα μειωθεί, ενώ με j η μεταβλητή που βρίσκεται πιο αριστερά – και άρα θα αυξηθεί.

Αν τελικά το μικρότερο λόγο έχει η r, θα μειωθεί κατά I_{res} αν το επιτρέπει το κάτω όριό της. Στην αντίθετη περίπτωση, θα μειωθεί όσο είναι δυνατόν, θα ξαναβρεθεί το νέο I_{res} και θα ξανασυγκριθούν οι λόγοι, ώστε να συνεχιστεί η επανάκτηση του I_{res}. Αν έχει μικρότερο λόγο ένας συνδυασμός μεταβλητών x_i, τότε το πόσο θα μειωθούν/ αυξηθούν εξαρτάται από τους συντελεστές τους στον πρώτο περιορισμό και θα είναι $(V_k - V_j)\delta = |I_{res}|$, όπου δ η αντίστοιχη μεταβολή στις μεταβλητές. Αν πάλι παραβιάζεται το άνω ή κάτω όριο μιας μεταβλητής, η μείωση/ αύξηση θα γίνει μέχρι το όριο αυτό, θα ξαναβρεθεί το I_{res} και θα ξανασυγκριθούν οι λόγοι. Η διαδικασία τελειώνει όταν το I_{res} γίνει μηδέν.

Κατά την εύρεση μεταβλητών για μείωση, υπάρχει περίπτωση καμιά να μην πληροί τις προϋποθέσεις. Σε αυτή την περίπτωση ο κόμβος του δέντρου δεν έχει εφικτή λύση, διαγράφεται και συνεχίζεται η επίλυση άλλων.

Θετικό I_{res} και T_{res}

Παρόμοιες κινήσεις θα γίνουν και σε αυτή την περίπτωση με τη διαφορά όμως ότι τώρα υπάρχει περίσσευμα αποθέματος και χρόνου να κατανεμηθεί, άρα θα επιλέγονται μεταβλητές για αύξηση πιο δεξιά στη λίστα.

Όταν και οι δύο περιορισμοί έχουν δεξί μέλος μεγαλύτερο του μηδενός, πρέπει – όπως και παραπάνω, να διορθωθεί πρώτα ο δεύτερος περιορισμός. Έτσι αναζητείται η μεταβλητή x_i (η r δεν συμμετέχει στο δεύτερο περιορισμό), η τιμή της οποίας βρίσκεται κάτω από το άνω όριό της και έχει τον μεγαλύτερο λόγο $\frac{R_i}{V_i}$ εκ των υπολοίπων. Το πόσο θα μεταβληθεί εξαρτάται από το άνω όριό της, έτσι αν μπορεί να «απορροφήσει» όλο το T_{res}, τότε ο περιορισμός έχει ικανοποιηθεί και πρέπει να ακολουθήσει η διαδικασία διόρθωσης του πρώτου περιορισμού όπως περιγράφεται παρακάτω. Αντίθετα, η μεταβλητή θα αυξηθεί όσο το δυνατόν περισσότερο, θα ξαναυπολογιστεί το T_{res} και θα αναζητηθεί ξανά μεταβλητή x_i για αύξηση.

Θετικό I_{res}

Στην περίπτωση που το δεξί μέλος του περιορισμού αποθέματος είναι θετικό πρέπει να βρεθεί η μεταβλητή στην οποία θα κατανεμηθεί. Σε αντίθεση όμως με το αρνητικό I_{res}, η κίνηση στη λίστα γίνεται προς τα δεξιά. Οι επιλογές για το ποια μεταβλητή θα αυξηθεί

αξιολογούνται με βάση τους λόγους τους. Για την r, ο λόγος είναι C. Για τις x_i, θα πρέπει να γίνει ταυτόχρονη αύξηση και μείωση ώστε να διατηρηθεί το T_{res} στο μηδέν. Πρέπει δηλαδή να βρεθεί μία μεταβλητή με θετική τιμή, μεγαλύτερη του κάτω ορίου της για να μειωθεί και μία που να βρίσκεται πιο δεξιά της πρώτης στη λίστα για να αυξηθεί. Για κάθε συνδυασμό θα βρεθεί ο λόγος $\frac{R_k - R_j}{V_k - V_j}$, όπου k είναι η μεταβλητή που βρίσκεται πιο δεξιά στη λίστα και άρα θα αυξηθεί και j η μεταβλητή που βρίσκεται πιο αριστερά στη λίστα και άρα θα μειωθεί. Η βέλτιστη κίνηση για την κατανομή του περίσσιου αποθέματος είναι αυτή που αντιστοιχεί στο μεγαλύτερο λόγο.

Αν η r έχει το μεγαλύτερο λόγο, θα αυξηθεί ανεξάρτητα από τις άλλες μεταβλητές προσέχοντας μόνο να μην υπερβεί το άνω όριό της. Η τιμή της θα αυξηθεί κατά I_{res} (διότι ο συντελεστής της στο πρώτο περιορισμό είναι 1) αν γίνεται ή όσο της επιτρέπει το όριό της. Αν τελικά το I_{res} δεν γίνει μηδέν με αυτή την κίνηση, πρέπει να βρεθεί εκ νέου μεταβλητή για αύξηση. Αν τελικά τον μεγαλύτερο λόγο έχει ένας συνδυασμός x_i, τότε η μείωση και αύξηση κάθε μιας εξαρτάται από τους συντελεστές τους στο πρώτο περιορισμό και θα είναι $(V_k - V_j)\delta = I_{res}$, όπου δ η αντίστοιχη μεταβολή στις μεταβλητές. Ομοίως με την περίπτωση της r, αν λόγω κάποιου άνω ή κάτω ορίου δεν μηδενιστεί το I_{res}, ξαναυπολογίζονται οι λόγοι.

Όπως και στην περίπτωση του αρνητικού I_{res}, υπάρχει η πιθανότητα να μην υπάρχει διαθέσιμη μεταβλητή για αύξηση. Τότε το πρόβλημα θεωρείται ότι έχει τελειώσει και έχει βρεθεί η βέλτιστη λύση παρά το μη μηδενικό υπόλοιπο του πρώτου περιορισμού.

Όπως και στην κλασική μέθοδο branch and bound, υπάρχουν τρεις λόγοι για να μη συνεχιστεί η διακλάδωση από έναν κόμβο του δέντρου:

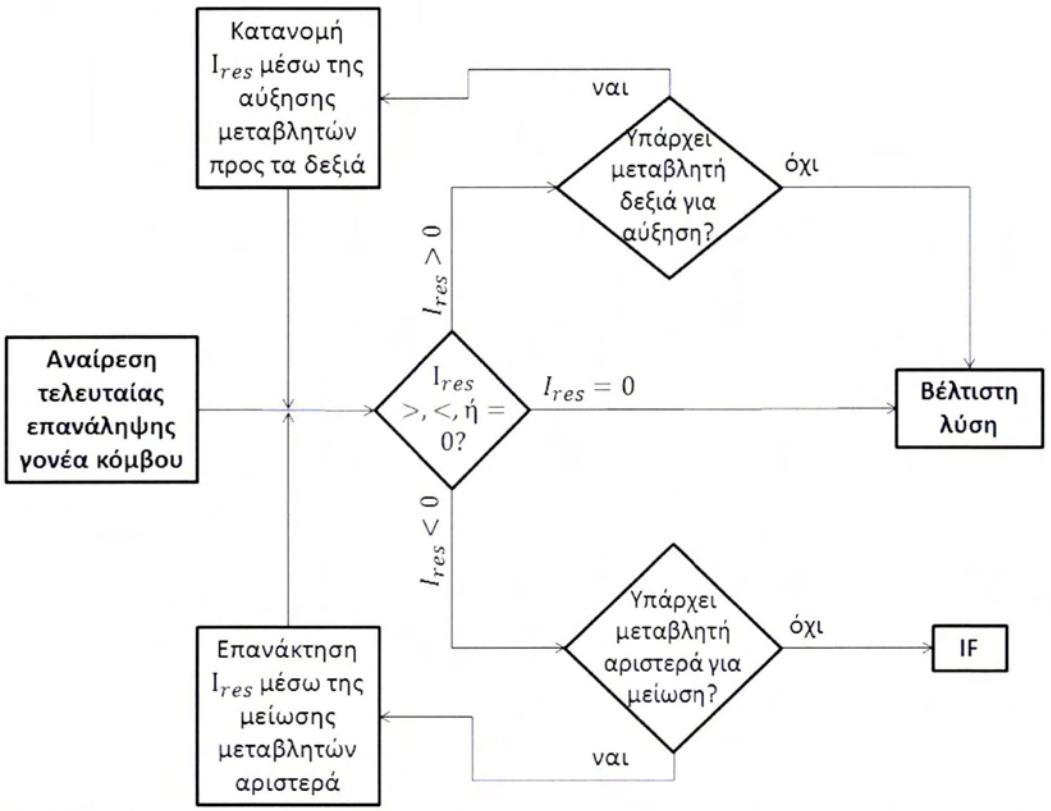
- Δεν έχει εφικτή λύση
- Έχει ως αποτέλεσμα ακέραια λύση
- Έχει ως αποτέλεσμα μη ακέραια λύση με τιμή αντικειμενικής χειρότερη από αυτή μιας ακέραιας λύσης

4.2 Νέα μεθοδολογία επίλυσης

Συμπληρώνοντας την μέθοδο που περιγράφηκε παραπάνω, εδώ θα γίνει πιο ξεκάθαρη η χρησιμότητα της λίστας μεταβλητών, καθώς τώρα θα γίνει γρηγορότερα η επίλυση των κόμβων, όπως φαίνεται στο διάγραμμα 2, μετά την εύρεση της λύσης της γραμμικής χαλάρωσης, πρώτος κόμβος του δέντρου branch and bound. Η ανάπτυξη του δέντρου

γίνεται με την επιβολή άνω και κάτω ορίου στην τελευταία μη ακέραια μεταβλητή που μπήκε στη λύση στον κόμβο γονέα, είτε είναι κάποια από τις x_i είτε είναι η r. Η επιβολή ενός ορίου σε μια μεταβλητή σημαίνει ότι αρχικά η μεταβλητή θα πάρει την τιμή του ορίου.

Παρακάτω αναλύονται οι δυο τρόποι ανάπτυξης του δέντρου branch and bound, με βάση το είδος του ορίου που επιβάλλεται, άνω ή κάτω.



Διάγραμμα 2 **Διάγραμμα ροής για την επίλυση του κάθε κόμβου του δέντρου branch and bound χρησιμοποιώντας τη νέα μεθοδολογία.**

4.2.1 Άνω όριο

Βήμα 1ο: Μεταβολή τελευταίας επανάληψης – Επιβολή ορίου

Με την επιβολή άνω ορίου σε μια μεταβλητή, είτε αν πρόκειται για x_i είτε για r , η τιμή της μειώνεται κατά ένα ποσό δ. Το ποσό δ είναι ουσιαστικά η διαφορά της μη ακέραιης λύσης της μεταβλητής που «ήρθε» από τον κόμβο γονέα και της ακέραιας τιμής της λύσης με στρογγυλοποίηση προς τα κάτω. Εάν από τον κόμβο γονέα το υπόλοιπο του δεύτερου περιορισμού είναι θετικό και η μεταβληθείσα μεταβλητή είναι η x_i τότε ο αλγόριθμος μεταβάλλει τα υπόλοιπα και των δύο περιορισμών, δηλαδή τα I_{res} και T_{res}, ενώ αν η μεταβληθείσα μεταβλητή είναι η r τότε ο αλγόριθμος μεταβάλλει το υπόλοιπο του πρώτου περιορισμού, δηλαδή το I_{res}, και συνεχίζει στο βήμα 2.

Εάν όμως η επιβολή άνω ορίου γίνεται σε μεταβλητή x_i και από την προσωρινή λύση της προηγούμενης παραγράφου υπάρχει θετικό υπόλοιπο αποθέματος, δηλαδή $I_{res} > 0$, τότε ο αλγόριθμος δίνει το ποσό δ στη μεταβλητή $x_j, (j \neq i)$ η οποία ήταν μέρος της τελευταίας λύσης, μεταβάλλοντας έτσι τα υπόλοιπα των δυο περιορισμών, δηλαδή τα I_{res} και T_{res}. Αν δεν μπορεί η μεταβλητή να δεχθεί αυτή την αλλαγή τότε απλώς μεταβάλλει τα υπόλοιπα των δυο περιορισμών και η x_i διατηρεί τιμή ίση με το άνω όριο της. Αν όμως υπάρχει μια τέτοια μεταβλητή την αυξάνει κατά δ.

Αν από τη διαδικασία του 1ου βήματος προκύψει μηδενικό διαθέσιμο απόθεμα, δηλαδή $I_{res} = 0$, ή/ και η μεταβλητή που της επιβλήθηκε άνω όριο ήταν η τελευταία στη λίστα τότε η λύση που βρέθηκε παραπάνω είναι η βέλτιστη για τον συγκεκριμένο κόμβο. Διαφορετικά ο αλγόριθμος πηγαίνει στο 2ο βήμα.

Βήμα 2ο: Επίλυση κόμβου

Σε αυτό το βήμα ο αλγόριθμος προσπαθεί να δώσει το διαθέσιμο απόθεμα στο σύστημα προκειμένου να βελτιωθεί η τιμή της αντικειμενικής και σταματάει όταν φτάσει στο τέλος της λίστας ή/ και όταν δεν υπάρχει άλλο διαθέσιμο απόθεμα. Ουσιαστικά θα αυξηθεί [1] η τιμή κάποιας μεταβλητής δεξιότερα της x_i.

Αναλυτικότερα, αν η μεταβλητή που περιορίστηκε από άνω όριο είναι η r ή αν είναι κάποια μεταβλητή x_i και το υπόλοιπο του δεύτερου περιορισμού είναι θετικό τότε η πρώτη μεταβλητή που βρίσκεται δεξιότερα στη λίστα θα πάρει το διαθέσιμο υπόλοιπο. Φυσικά αντιστοίχως θα μεταβληθούν και τα υπόλοιπα των δυο περιορισμών.

Στην περίπτωση όπου η μεταβλητή που περιορίστηκε από άνω όριο είναι κάποια από τις x_i και ο δεύτερος περιορισμός είναι ίσος με το μηδέν τότε θα πρέπει να βρεθεί μια μεταβλητή αριστερά της x_{i+1} που να μπορεί να μειωθεί[2], συμπεριλαμβανομένης και της x_i[3]. Αυτό γίνεται ώστε να εξισορροπηθεί η αύξηση μεταβλητής δεξιά της x_i, δηλαδή της μεταβλητής x_{i+1}. Εάν δεν υπάρχει καμία μεταβλητή που μπορεί να μειωθεί τότε η λύση που βρέθηκε στο 1ο βήμα θα αποτελεί τη βέλτιστη για αυτόν τον κόμβο. Εάν υπάρχει μεταβλητή τότε αυτή θα μειωθεί κατά το μέγιστο δυνατό, δηλαδή είτε θα μειωθεί τόσο ώστε να μηδενισθεί

[1] Μια μεταβλητή θεωρείται ότι μπορεί να αυξηθεί όταν δεν έχει φτάσει στο άνω όριο της, δηλαδή είναι μικρότερη από αυτό.

[2] Μια μεταβλητή θεωρείται ότι μπορεί να μειωθεί όταν δεν έχει φτάσει στο κάτω όριο της, δηλαδή είναι μεγαλύτερη από αυτό.

[3] Με την προηγούμενη μεθοδολογία η αναζήτηση μιας μεταβλητής προς μείωση θα ξεκινούσε από την αρχή της λίστας με τη χρήση των λόγων που προαναφέρθηκαν, πράγμα το οποίο θα ήταν πιο χρονοβόρο.

το διαθέσιμο απόθεμα ($I_{res} = 0$) βάση του ποσού δ είτε μέχρι εκεί που επιτρέπει το κάτω όριο της. Αυτό σημαίνει ότι αν η μειωμένη μεταβλητή πάρει την τιμή της βάση του κάτω ορίου της τότε η αυξανόμενη, δηλαδή η x_{i+1}, θα πάρει τη νέα τιμή με βάση το ποσό δ, ενώ αν η μειούμενη πάρει την τιμή βάσει του ποσού δ τότε η αυξανόμενη θα την πάρει είτε βάσει του άνω ορίου είτε βάσει του ποσού δ.

Το βήμα 2 θα επαναληφθεί μέχρι να βρεθεί η βέλτιστη τιμή για αυτόν τον κόμβο.

4.2.2 Κάτω όριο

Βήμα 1ο: Μεταβολή τελευταίας επανάληψης – Επιβολή ορίου

Με την επιβολή κάτω ορίου σε μια μεταβλητή, είτε σε κάποια από τις x_i είτε στην r, η τιμή της αυξάνεται κατά ένα ποσό δ, όπως και με την επιβολή άνω ορίου. Το ποσό δ εδώ είναι η διαφορά της μη ακέραιης λύσης της μεταβλητής που «ήρθε» από τον κόμβο γονέα και της ακέραιας τιμής της λύσης με στρογγυλοποίηση προς τα πάνω, κατά απόλυτη τιμή. Εάν από τον κόμβο γονέα το υπόλοιπο του δεύτερου περιορισμού, χρονικός περιορισμός, είναι θετικό τότε ο αλγόριθμος μεταβάλλει τα υπόλοιπα και των δύο περιορισμών, δηλαδή τα I_{res} και T_{res}, ενώ αν η μεταβληθείσα μεταβλητή είναι η r τότε ο αλγόριθμος μεταβάλλει το υπόλοιπο του πρώτου περιορισμού, δηλαδή το I_{res}, και συνεχίζει στο βήμα 2.

Εάν όμως η επιβολή κάτω ορίου γίνεται σε μεταβλητή x_i και από την προσωρινή λύση της προηγούμενης παραγράφου υπάρχει αρνητικό υπόλοιπο αποθέματος, δηλαδή $I_{res} < 0$, τότε ο αλγόριθμος αφαιρεί το ποσό δ από την πρώτη μεταβλητή η οποία βρίσκεται πιο αριστερά στη λίστα από τη x_i και η οποία θα μπορεί να μειωθεί, μεταβάλλοντας έτσι τα υπόλοιπα των δυο περιορισμών, δηλαδή τα I_{res}και T_{res}. Αν δεν υπάρχει τέτοια μεταβλητή τότε ο κόμβος δεν έχει εφικτή λύση οπότε το δέντρο δεν αναπτύσσεται από αυτό το σημείο και μετά. Αν υπάρχει μεταβλητή να μειωθεί τότε μειώνει την τιμή της κατά δ.

Αν από τη διαδικασία του 1ου βήματος προκύψει μηδενικό διαθέσιμο απόθεμα, δηλαδή $I_{res} = 0$, ή/ και η μεταβλητή που της επιβλήθηκε κάτω όριο ήταν η τελευταία στη λίστα τότε η λύση που βρέθηκε παραπάνω είναι η βέλτιστη για τον συγκεκριμένο κόμβο. Διαφορετικά ο αλγόριθμος πηγαίνει στο 2ο βήμα, όπως ακριβώς και στην περίπτωση της επιβολής άνω ορίου.

Βήμα 2ο: Επίλυση κόμβου

Σε αυτό το βήμα ο αλγόριθμος προσπαθεί να «αφαιρέσει» το διαθέσιμο απόθεμα από το σύστημα γιατί έχει δώσει περισσότερο από το μέγιστο που έχει στη διάθεσή του και

σταματάει όταν φτάσει στην αρχή της λίστας ή/ και όταν το διαθέσιμο απόθεμα πάψει να είναι αρνητικό. Ουσιαστικά θα μειωθεί η τιμή κάποιας μεταβλητής αριστερότερα από τη x_i. Πιο συγκεκριμένα, αν η μεταβλητή που περιορίστηκε από κάτω όριο είναι η r ή αν είναι κάποια μεταβλητή x_i και το υπόλοιπο του δεύτερου περιορισμού είναι θετικό τότε η πρώτη μεταβλητή που βρίσκεται αριστερά στη λίστα θα χάσει το πλασματικό απόθεμα, αυτό δηλαδή που έχει πάρει το σύστημα χωρίς πραγματικά να το έχει στη διάθεσή του. Φυσικά αντιστοίχως θα μεταβληθούν και τα υπόλοιπα των δυο περιορισμών.

Στην περίπτωση που η μεταβλητή που περιορίστηκε από κάτω όριο είναι κάποια από τις μεταβλητές x_i και ο δεύτερος περιορισμός είναι ίσος με το μηδέν (ο πρώτος περιορισμός είναι αρνητικός) τότε θα πρέπει να βρεθεί μια μεταβλητή αριστερά της x_{i-1} που να μπορεί να αυξηθεί. Αν υπάρχει μια τέτοια μεταβλητή τότε αυτή θα αυξηθεί κατά τον μέγιστο δυνατό τρόπο, με ανάλογη διαδικασία με αυτή που αναφέρθηκε στον κλάδο του άνω ορίου, δηλαδή η αύξηση θα γίνει είτε μέχρι εκεί που της επιτρέπει το άνω όριό της, είτε μέχρις ότου να μηδενισθεί το αρνητικό απόθεμα. Αν όμως δεν υπάρχει μεταβλητή για αύξηση τότε αλγόριθμος θα αναζητήσει μεταβλητή για μείωση ξεκινώντας από την x_{i-1} και πηγαίνοντας προς τα αριστερά. Εάν βρεθεί κάποια μεταβλητή τότε θα τη μειώσει όσο χρειάζεται βάση του ποσού δ, ενώ αν δεν υπάρχει τότε ο κόμβος αυτός δεν έχει εφικτή λύση οπότε η ανάπτυξη του δέντρου δεν συνεχίζει σε αυτόν τον κλάδο.

4.3 Αριθμητικό παράδειγμα

4.3.1 Δεδομένα παραδείγματος

Για την περαιτέρω κατανόηση της θεωρίας του αλγορίθμου αναλύεται παρακάτω ένα αριθμητικό παράδειγμα.

$$Max\ 4x_1 + 7x_2 + 9x_3 + 10x_4 + 2r$$

$$s.t \quad x_1 + 2x_2 + 4x_3 + 6x_4 + r \leq 33.5$$

$$x_1 + x_2 + x_3 + x_4 \leq 10$$

Πριν ξεκινήσει η επίλυση του προβλήματος θα πρέπει να εξαχθούν οι πληροφορίες που χρειάζονται. Οι πληροφορίες αυτές είναι οι παράμετροι του προβλήματος και συγκεκριμένα τα κόστη (R_i), οι όγκοι των πωλήσεων (V_i), η τιμή πώλησης στο τέλος του χρονικού ορίζοντα (C), το αρχικό διαθέσιμο απόθεμα (I) και οι συνολικοί χρονικοί περίοδοι (T). Οι τιμές των πληροφοριών φαίνονται στους ακόλουθους πίνακες:

Πίνακας 1 **Παράμετροι του προβλήματος στην αντικειμενική συνάρτηση.**

R_1	R_2	R_3	R_4	C
4	7	9	10	2

Πίνακας 2 **Παράμετροι του προβλήματος στους δυο περιορισμούς.**

V_1	V_2	V_3	V_4	I	T
1	2	4	6	33.5	10

4.3.2 Δημιουργία λίστας

Το πρώτο βήμα του αλγορίθμου είναι η δημιουργία της λίστας για την κατάταξη των μεταβλητών απόφασης βάσει κάποιων συγκεκριμένων λόγων.

Πρώτη μεταβλητή

Για το πρώτο στοιχείο χρησιμοποιείται ο λόγος των συντελεστών των μεταβλητών x_i, δηλαδή $\frac{R_i}{V_i}$, ενώ για τη μεταβλητή r θα χρησιμοποιείται ως λόγος η παράμετρος C για τη σύγκριση με τους λόγους των x_i. Οι τιμές των λόγων φαίνονται στον πίνακα 3.

Πίνακας 3 **Οι λόγοι των μεταβλητών x_i και ο λόγος της r για την εύρεση του πρώτου στοιχείου της λίστας.**

μεταβλητές	x_1	x_2	x_3	x_4	r
λόγοι $\frac{R_i}{V_i}$	4	$\frac{7}{2} = 3.5$	$\frac{9}{4} = 2.25$	$\frac{10}{6} = 1.67$	2

Από τον πίνακα 3 γίνεται αντιληπτό ότι το πρώτο στοιχείο στη λίστα θα είναι η μεταβλητή x_1, γιατί έχει τον μεγαλύτερο λόγο. Οπότε $L = \{x_1\} = \{4\}$.

Δεύτερη μεταβλητή

Στη συνέχεια αναζητείται το δεύτερο στοιχείο. Για το δεύτερο και κάθε επόμενο στοιχείο θα χρησιμοποιείται πλέον ο λόγος των μεταβλητών x_i, ο οποίος εξαρτάται από την προηγούμενη μεταβλητή που μπήκε στη λίστα. Συγκεκριμένα, για τη δεύτερη μεταβλητή ο λόγος είναι $\frac{R_i - R_1}{V_i - V_1}$.Οι τιμές των λόγων για τη δεύτερη μεταβλητή που θα μπει στη λίστα δίνονται στον πίνακα 4 παρακάτω.

Πίνακας 4 **Οι λόγοι των μεταβλητών x_i, που δεν έχουν εισέλθει στη λίστα σε σχέση με τη x_1.**

μεταβλητές	x_2	x_3	x_4
λόγοι $\frac{R_i - R_1}{V_i - V_1}$	$\frac{7-4}{2-1} = 3$	$\frac{9-4}{4-1} = 1.67$	$\frac{10-4}{6-1} = 1.2$

Ο μεγαλύτερος λόγος είναι αυτός της x_2 οπότε αυτή θα είναι το επόμενοι στοιχείο που θα εισαχθεί στη λίστα. Οπότε η λίστα μέχρι τώρα είναι $L = \{x_1, x_2\} = \{4, 3\}$.

Τρίτη μεταβλητή

Για την τρίτη μεταβλητή που θα μπει στη λίστα θα χρησιμοποιηθεί πάλι ο λόγος $\frac{R_i - R_1}{V_i - V_1}$ των μεταβλητών x_i, ο οποίος όμως τώρα θα εξαρτάται από την τελευταία μεταβλητή που εισήχθη δηλαδή την x_2 οπότε θα έχει τη μορφή $\frac{R_i - R_2}{V_i - V_2}$. Στον πίνακα 5 φαίνονται οι τιμές των λόγων.

Πίνακας 5 **Οι λόγοι των μεταβλητών x_i, που δεν έχουν εισέλθει στη λίστα σε σχέση με τη x_2, η οποία εισήλθε τελευταία.**

μεταβλητές	x_3	x_4
λόγοι $\frac{R_i - R_2}{V_i - V_2}$	$\frac{9-7}{4-2} = 1$	$\frac{10-7}{6-2} = 0.75$

Το τρίτο στοιχείο της λίστας θα είναι η μεταβλητή x_3, οπότε η λίστα γίνεται $L = \{x_1, x_2, x_3\} = \{4, 3, 1\}$.

Τέταρτη μεταβλητή

Η τέταρτη μεταβλητή θα είναι προφανώς η εναπομένουσα x_4. Παρόλα αυτά είναι σημαντικό να βρεθεί ο λόγος προκειμένου να μπορεί και η μεταβλητή r να εισέλθει στη σωστή θέση στη λίστα. Η τιμή του λόγου της x_4 υπολογίζεται από τον τύπο $\frac{R_4 - R_3}{V_4 - V_3}$ και εξαρτάται από τη x_3, οπότε η τιμή της είναι $\frac{10-9}{6-4} = 0.5$. Τελικά η λίστα για τις μεταβλητές x είναι $L = \{x_1, x_2, x_3, x_4\} = \{4, 3, 1, 0.5\}$.

Μεταβλητή r

Τέλος, θα πρέπει και η r να εισαχθεί στην κατάλληλη θέση στη λίστα. Η r έχει λόγο ίσο με 2 και άρα θα μπει στην τρίτη θέση, δηλαδή μετά την x_2και πριν την x_3. Οπότε η τελική μορφή της λίστας, που θα περιλαμβάνει όλες τις μεταβλητές, θα είναι $L = \{x_1, x_2, r, x_3, x_4\} = \{4, 3, 2, 1, 0.5\}$.

4.3.3 Εύρεση πρώτης λύσης

Για την εύρεση της γραμμικής χαλάρωσης του παραπάνω προβλήματος που θα αποτελέσει και τον πρώτο κόμβο του δέντρου branch and bound, ακολουθείται η παρακάτω διαδικασία:

Πρώτο βήμα

Ξεκινώντας από το πρώτο στοιχείο - μεταβλητή της λίστας αναζητείται η καλύτερη τιμή που μπορεί να λάβει και η οποία δεν θα παραβιάζει κανέναν από τους δυο περιορισμούς. Από τον πρώτο περιορισμό η τιμή είναι 33.5, ενώ από το δεύτερο είναι 10, δηλαδή $x_1 = min\{33.5, 10\}$. Άρα το x_1 θα πάρει την τιμή 10 ($x_1 = 10$) από τον 2° περιορισμό. Οι υπόλοιπες μεταβλητές θα έχουν την τιμή 0. Στο σημείο αυτό γίνεται έλεγχος του 1ου περιορισμού και προκύπτει ότι $I_{res} = 23.5$. Επειδή υπάρχει διαθέσιμο απόθεμα, δηλαδή $I_{res} > 0$, που μπορεί να δοθεί στο σύστημα για να βελτιωθεί η τιμή της αντικειμενικής αυτό σημαίνει ότι θα υπάρξει και άλλο βήμα ως ότου το διαθέσιμο απόθεμα γίνει μηδέν. Συγκεντρωτικά, τα αποτελέσματα του πρώτου βήματος είναι:

Πίνακας 6 **Αποτελέσματα της πρώτης επανάληψης για την εύρεση της πρώτης λύσης.**

$x_1 = 10$	$I_{res} = 23.5$
$x_2 = x_3 = x_4 = r = 0$	$T_{res} = 0$
	$z^* = 40$

Δεύτερη επανάληψη

Στη δεύτερη επανάληψη ο αλγόριθμος κινείται προς το δεύτερο στοιχείο - μεταβλητή της λίστας, το οποίο είναι το x_2. Με την ίδια λογική με προηγουμένως αναζητείται η βέλτιστη τιμή που μπορεί να πάρει αυτή η μεταβλητή. Από τον πρώτο περιορισμό η μέγιστη δυνατή τιμή είναι $\frac{I_{res}}{V_2 - V_1} = \frac{23.5}{2-1} = 11.75$, ενώ από το δεύτερο είναι 10, δηλαδή $x_2 = \min\{11.75, 10\}$. Παρότι ο 2ος περιορισμός έχει ικανοποιηθεί λαμβάνεται υπόψη επειδή η μεταβλητή x_2 περιορίζεται από αυτόν αλλά η τιμή της x_1 μπορεί να μεταβληθεί αναλόγως αν αυτό βελτιώνει τη λύση, δηλαδή η τιμή που θα πάρει η x_2 θα οδηγήσει σε αντίστοιχη μείωση την τιμή της x_1. Εν τέλει είναι προφανές ότι $x_2 = 10$, άρα λόγω του 2ου περιορισμού $x_1 = T - x_2 = 10 - 10 = 0$. Αφού βρέθηκε η τιμή της δεύτερης μεταβλητής γίνεται έλεγχος του διαθέσιμου αποθέματος και προκύπτει ότι $I_{res} = 13.5 > 0$, άρα θα χρειαστεί και άλλη επανάληψη. Τα αποτελέσματα της δεύτερης επανάληψης δίνονται στον πίνακα 7.

Πίνακας 7 **Αποτελέσματα της δεύτερης επανάληψης για την εύρεση της πρώτης λύσης.**

$x_2 = 10$	$I_{res} = 13.5$
$x_1 = x_3 = x_4 = r = 0$	$T_{res} = 0$
	$z^* = 70$

Όπως ήταν αναμενόμενο, η τιμή της αντικειμενικής αυξήθηκε.

Τρίτη επανάληψη

Στην τρίτη επανάληψη επιλέγεται το 3° στοιχείο - μεταβλητή, η οποία είναι η r. Η r περιορίζεται μόνο από τον 1° περιορισμό και άρα η τιμή που θα πάρει δεν θα επηρεάσει τις μεταβλητές x. Αυτό σημαίνει ότι η βέλτιστη τιμή της θα εξαρτάται μόνο από αυτόν και άρα η τιμή που θα πάρει είναι το υπόλοιπο διαθέσιμο απόθεμα, δηλαδή $r = I_{res} = 13.5$, και έτσι μηδενίζεται το απόθεμα και βρίσκεται η πρώτη λύση η οποία δίνεται στον πίνακα 8.

Πίνακας 8 **Αποτελέσματα της τρίτης επανάληψης για την εύρεση της γραμμικής χαλάρωσης.**

$x_2 = 10$	$I_{res} = 0$
$r = 13.5$	$T_{res} = 0$
$x_1 = x_3 = x_4 = 0$	$z^* = 97$

Από τη στιγμή που επιλέχθηκε η r ως μεταβλητή για να μπει στη βάση ήταν αναμενόμενο ότι οι επαναλήψεις θα τερματιστούν εδώ και σε αυτή την επανάληψη θα βρισκόταν η τελική λύση. Αυτό συμβαίνει γιατί η r δεν εμφανίζεται στον 2° περιορισμό και άρα όποια μεταβολή της, η οποία αναμφίβολα θα σχετίζεται με τον 1°, δεν θα επηρεάσει τις υπόλοιπες μεταβλητές x δεδομένου ότι υπάρχει διαθέσιμο απόθεμα.

Συγκεντρωτικά τα βήματα

Πίνακας 9 **Συγκεντρωτικά οι επαναλήψεις για την εύρεση της τελικής πρώτης λύσης στο 3ο βήμα - επανάληψη.**

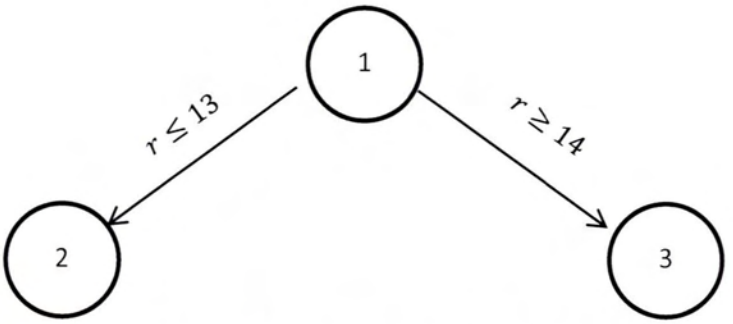
1° βήμα	2° βήμα	3° βήμα
$x_1 = 10$ $I_{res} = 23.5$ $T_{res} = 0$ $z^* = 40$	$x_2 = 10$ $I_{res} = 13.5$ $T_{res} = 0$ $z^* = 70$	$x_2 = 10$ $r = 13.5$ $I_{res} = 0$ $T_{res} = 0$ $z^* = 97$

4.3.4 Δέντρο Branch and Bound

Μετά την εύρεση της πρώτης λύσης ξεκινούν να εισάγονται ακέραια όρια σε κάποια μεταβλητή που δεν έχει ακέραια τιμή.

Γονικός κόμβος 1

Στο συγκεκριμένο παράδειγμα η μόνη μη ακέραια μεταβλητή είναι η r με τιμή 13.5 και άρα σε αυτή θα εφαρμοστούν δύο όρια, $r \leq 13$ και $r \geq 14$, όπως φαίνεται στο σχήμα 1.



Σχήμα 1 **Οι δυο πρώτοι κόμβοι που σχηματίζονται με την εφαρμογή ορίων στην πρώτη λύση του χαλαρού προβλήματος.**

Κόμβος 2

Στον κόμβο 2 η r παίρνει την τιμή 13 αρχικά, βάσει του ορίου που εφαρμόσθηκε. Στην αρχή της εύρεση της βέλτιστης λύσης του κόμβου 2 οι τιμές των μεταβλητών και των παραμέτρων φαίνονται στον πίνακα 10.

Πίνακας 10 **Οι τιμές των μεταβλητών και των παραμέτρων στην αρχή της επίλυσης του κόμβου 2.**

$x_2 = 10$	$I_{res} = 0.5$
$r = 13 \leq 13$	$T_{res} = 0$
$x_1 = x_3 = x_4 = 0$	$z^* = 96$

Όπως φαίνεται το απόθεμα έχει πάρει θετική τιμή, $I_{res} > 0$, αυτό σημαίνει ότι υπάρχει απόθεμα διαθέσιμο να δοθεί στο σύστημα προκειμένου να αυξηθεί η τιμή της αντικειμενικής, δηλαδή τα συνολικά έσοδα. Για να γίνει το διαθέσιμο απόθεμα 0 θα πρέπει να αυξηθεί η τιμή κάποιας μεταβλητής. Όμως αν η μεταβλητή αυτή είναι κάποια από τις x, τότε θα πρέπει κάποια άλλη μεταβλητή x να μειωθεί ώστε να μην παραβιαστεί ο δεύτερος περιορισμός. Παρακάτω γίνεται η επίλυση του κόμβου χρησιμοποιώντας και τις δυο μεθοδολογίες για σύγκριση.

Αρχική μεθοδολογία

Με την αρχική μεθοδολογία για την εύρεση των δυο μεταβλητών που θα μεταβληθούν πρέπει να υπολογισθούν οι λόγοι $\frac{R_k - R_j}{V_k - V_j}$ όλων των μεταβλητών x, όπου k είναι η μεταβλητή που βρίσκεται πιο δεξιά στη λίστα και άρα θα αυξηθεί και j η μεταβλητή που βρίσκεται πιο αριστερά στη λίστα και άρα θα μειωθεί. Για την r ο λόγος είναι σταθερός και ίσος με 2, ο συντελεστής κέρδους του δηλαδή. Στη συνέχεια όλοι οι λόγοι θα έπρεπε να συγκριθούν και να επιλεγεί ο μεγαλύτερος λόγος. Οι συνδυασμοί και οι τιμές των λόγων των μεταβλητών δίνονται στον πίνακα 11 της επόμενης σελίδας.

Πίνακας 11 **Οι δυνατοί συνδυασμοί και οι αντίστοιχοι λόγοι των μεταβλητών για εύρεση του βέλτιστου συνδυασμού για τον κόμβο 2.**

α/α	Μεταβλητές		Τιμές των λόγων τους
	Προς αύξηση (k)	Προς μείωση (j)	
1	x_2	x_1	3
2	x_3	x_1	$\frac{5}{3} \cong 1.667$
3	x_3	x_2	1
4	x_4	x_1	$\frac{6}{5} = 1.2$
5	x_4	x_2	$\frac{3}{4} = 0.75$
6	x_4	x_3	$\frac{1}{2} = 0.5$
7	r	-	2

Από τους παραπάνω συνδυασμούς κάποιοι δεν είναι εφικτοί λόγω αδυναμίας μεταβολής των αντίστοιχων μεταβλητών. Πιο συγκεκριμένα, όσοι συνδυασμοί έχουν τις μεταβλητές x_1 και x_3 προς μείωση δεν είναι πραγματοποιήσιμοι διότι η x_1 και η x_3 βρίσκονται ήδη στο κάτω όριο τους οπότε δεν μπορούν να μειωθούν περισσότερο[4]. Επίσης, η μεταβλητή x_2 βρίσκεται ήδη στο άνω όριο της οπότε ο πρώτος συνδυασμός απορρίπτεται, όπως και η r. Οπότε από τους εναπομείναντες εφικτούς συνδυασμούς, δηλαδή τον 3° και τον 5°, επιλέγεται ο 3ος γιατί έχει τον μεγαλύτερο λόγο.

Η τιμή της αύξησης και της μείωσης για τις δυο μεταβλητές θα είναι η ίδια τιμή δ και θα καθοριστεί από τον 1° περιορισμό. Συγκεκριμένα, θα εξαρτάται από τους συντελεστές των μεταβλητών στον 1° περιορισμό, τους όγκους πωλήσεων δηλαδή, και θα υπολογιστεί από τον τύπο $(V_3 - V_2)\delta = I_{res}$, άρα $(4-2)\delta = 0.5 \rightarrow \delta = 0.25$. Οπότε οι νέες τιμές των x_2 και x_3 θα είναι αντίστοιχα 9.75 και 0.25.

Νέα μεθοδολογία

1. Ο αλγόριθμος αναζητά στη λίστα την πρώτη διαθέσιμη μεταβλητή δεξιά της r και η οποία μπορεί να αυξηθεί. Η μεταβλητή αυτή είναι η x_3. Επειδή η x_3 υπάρχει και στον 2° περιορισμό θα πρέπει κάποια μεταβλητή x να μειωθεί.
2. Τώρα ο αλγόριθμος κινείται αντίθετα, δηλαδή αναζητεί μια μεταβλητή x πιο αριστερά της x_3 η οποία θα μπορεί να μειωθεί, συμπεριλαμβανομένης και της r. Στη

[4] Ως κάτω όριο στην αρχή του προβλήματος λογίζεται για όλες τις μεταβλητές το 0, ενώ ως άνω για την r λογίζεται το αρχικό διαθέσιμο απόθεμα I και για τις x_i βρίσκεται από τη σχέση $min\{I/V_i, T\}$.

συγκεκριμένη περίπτωση η μόνη μεταβλητή που πληροί τις προϋποθέσεις αυτές είναι η x_2.

3. Η αύξηση και η μείωση των τιμών των δυο επιλεγέντων μεταβλητών θα γίνει όπως και στην αρχική μεθοδολογία. Οπότε οι νέες τιμές των x_2 και x_3 θα είναι αντίστοιχα 9.75 και 0.25.
4. Ελέγχονται τα υπόλοιπα των δυο περιορισμών και προκύπτει ότι και τα δυο είναι ίσα με το 0, $I_{res} = T_{res} = 0$, οπότε η λύση αυτή είναι η βέλτιστη για τον κόμβο 2.

Η λύση αυτή συμφωνεί με τη λύση που δίνεται από την αρχική μεθοδολογία, με τη διαφορά ότι εδώ η λύση βρίσκεται γρηγορότερα γιατί δεν χρειάζεται να ελεγχθούν όλες οι μεταβλητές, οπότε απαιτούνται λιγότερες πράξεις άρα και λιγότερος χρόνος.

Τα αποτελέσματα του κόμβου, ανεξαρτήτως μεθοδολογίας, δίνονται στον πίνακα 12.

Πίνακας 12 **Η βέλτιστη λύση για τον κόμβο 2.**

$x_2 = 9.75$	$I_{res} = 0$
$x_3 = 0.25$	$T_{res} = 0$
$r = 13 \leq 13$	
$x_1 = x_4 = 0$	$z^* = 96.5$

Όπως φαίνεται η τιμή της αντικειμενικής αυξήθηκε σε σχέση με αυτή που είχε στην αρχή της επίλυσης του κόμβου 2, αλλά μειώθηκε σε σχέση με την τιμή που είχε στον κόμβο 1.

Κόμβος 3

Στον κόμβο 3 η r παίρνει την τιμή 14 αρχικά, βάσει του ορίου που εφαρμόσθηκε. Στην αρχή της εύρεση της βέλτιστης λύσης του κόμβου 3 οι τιμές των μεταβλητών και των παραμέτρων φαίνονται στον πίνακα 13.

Πίνακας 13 **Οι τιμές των μεταβλητών και των παραμέτρων στην αρχή της επίλυσης του κόμβου 3.**

$x_2 = 10$	$I_{res} = -0.5$
$r = 14 \geq 14$	$T_{res} = 0$
$x_1 = x_3 = x_4 = 0$	$z^* = 98$

Το απόθεμα έχει πάρει αρνητική τιμή, $I_{res} < 0$, το οποίο σημαίνει ότι το σύστημα έχει πάρει περισσότερες μονάδες προϊόντος από τις διαθέσιμες οπότε θα πρέπει κάποια μεταβλητή να μειωθεί. Αν η μεταβλητή αυτή είναι κάποια από τις x, τότε κάποια άλλη x θα

πρέπει να αυξηθεί προκειμένου να μηδενισθεί και ο δεύτερος περιορισμός. Ακολουθεί η εύρεση της λύσης του κόμβου 3 χρησιμοποιώντας και τις δυο μεθόδους.

Αρχική μεθοδολογία

Με την αρχική μεθοδολογία για την εύρεση των δυο μεταβλητών που θα μεταβληθούν πρέπει να υπολογισθούν οι λόγοι $\frac{R_k - R_j}{V_k - V_j}$ όλων των μεταβλητών x (όπως έγινε στον κόμβο 2), όπου k είναι η μεταβλητή που βρίσκεται πιο δεξιά στη λίστα και άρα θα μειωθεί (σε αντίθεση με τον κόμβο 2) και j η μεταβλητή που βρίσκεται πιο αριστερά στη λίστα και άρα θα αυξηθεί (σε αντίθεση με τον κόμβο). Για την r ο λόγος είναι σταθερός και ίσος με 2. Στη συνέχεια όλοι οι λόγοι θα έπρεπε να συγκριθούν και να επιλεγεί ο μικρότερος λόγος. Οι συνδυασμοί και οι τιμές των λόγων τους δίνονται στον παρακάτω πίνακα.

Πίνακας 14 **Οι δυνατοί συνδυασμοί και οι αντίστοιχοι λόγοι των μεταβλητών για εύρεση του βέλτιστου συνδυασμού για τον κόμβο 3.**

α/α	Μεταβλητές		Τιμές των λόγων τους
	Προς μείωση (k)	Προς αύξηση (j)	
1	x_2	x_1	3
2	x_3	x_1	$\frac{5}{3} \cong 1.667$
3	x_3	x_2	1
4	x_4	x_1	$\frac{6}{5} = 1.2$
5	x_4	x_2	$\frac{3}{4} = 0.75$
6	x_4	x_3	$\frac{1}{2} = 0.5$
7	r	-	2

Από τους παραπάνω συνδυασμούς κάποιοι δεν είναι εφικτοί λόγω αδυναμίας μεταβολής των αντίστοιχων μεταβλητών. Αναλυτικότερα, όσοι συνδυασμοί έχουν τις μεταβλητές x_3 και x_4 προς μείωση δεν είναι πραγματοποιήσιμοι διότι η x_3 και η x_4 βρίσκονται ήδη στο κάτω όριο τους και δεν μπορούν να μειωθούν περισσότερο, όπως και η r. Επίσης, η μεταβλητή x_2 βρίσκεται ήδη στο άνω όριο και δεν μπορεί να αυξηθεί. Οπότε επιλέγεται ο πρώτος συνδυασμός γιατί μόνο αυτός είναι εφικτός.

Με παρόμοιο τρόπο με τον κόμβο 2 θα αυξηθούν και θα μειωθούν οι τιμές των δυο μεταβλητών. Συγκεκριμένα, ο τύπος που δίνει τη μεταβολή δ είναι $(V_1 - V_2)\delta = I_{res}$ και θα είναι $(1-2)\delta = -0.5 \rightarrow \delta = 0.5$. Άρα οι νέες τιμές των x_1 και x_2 αντίστοιχα θα είναι 0.5 και 9.5.

Νέα μεθοδολογία

1. Επειδή το απόθεμα έχει αρνητική τιμή ο αλγόριθμος αναζητά στη λίστα την πρώτη μεταβλητή που μπορεί να μειωθεί. Η μεταβλητή αυτή είναι η x_2, η οποία εμφανίζεται και στον 2° περιορισμό και άρα θα πρέπει κάποια άλλη μεταβλητή x να αυξηθεί.
2. Η μεταβλητή που θα αυξηθεί είναι η x_1 γιατί η πρώτη μεταβλητή x που βρίσκεται αριστερότερα από την x_2 και μπορεί να αυξηθεί.
3. Η μείωση και η αύξηση των τιμών των δυο επιλεγέντων μεταβλητών θα γίνει όπως και στην αρχική μεθοδολογία. Άρα οι νέες τιμές των x_1 και x_2 αντίστοιχα θα είναι 0.5 και 9.5.
4. Ελέγχονται οι δυο περιορισμοί και προκύπτει ότι και οι δυο είναι ίσοι με το 0 οπότε η λύση αυτή είναι η βέλτιστη για τον κόμβο 3.

Η λύση αυτή συμφωνεί με τη λύση που δίνεται από την αρχική μεθοδολογία, με τη διαφορά ότι εδώ η λύση βρίσκεται γρηγορότερα γιατί δεν χρειάζεται να ελεγχθούν όλες οι μεταβλητές, οπότε απαιτούνται λιγότερες πράξεις άρα και λιγότερος χρόνος.

Τα αποτελέσματα του κόμβου δίνονται στον πίνακα 15.

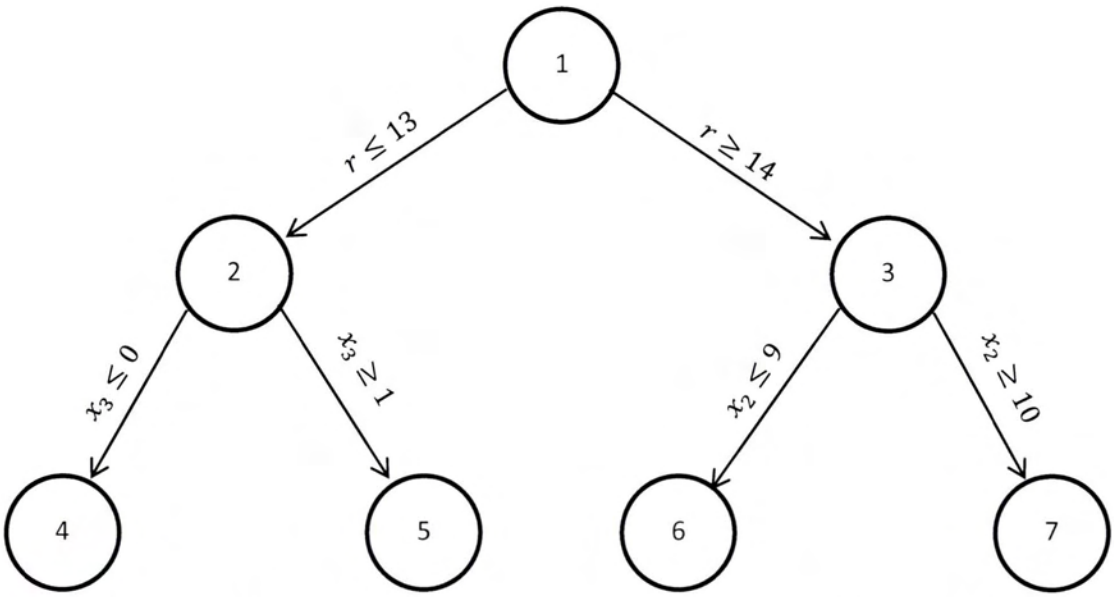
Πίνακας 15 **Η βέλτιστη λύση για τον κόμβο 3.**

$x_1 = 0.5$	$I_{res} = 0$
$x_2 = 9.5$	$T_{res} = 0$
$r = 14 \geq 14$	
$x_1 = x_4 = 0$	$z^* = 96.5$

Όπως παρατηρείται η τιμή της αντικειμενικής μειώθηκε σε σχέση με αυτή που είχε αμέσως μετά την εφαρμογή του ορίου στην r, το οποίο ήταν αναμενόμενο αφού αρχικά είχε δαπανηθεί περισσότερο απόθεμα από το διαθέσιμο. Παρόλα αυτά η τελική τιμή του κόμβου είναι μικρότερη από αυτή του κόμβου 1, το οποίο είναι επίσης λογικό και αναμενόμενο.

Γονικοί κόμβοι 2,3

Επειδή και στους δυο κόμβους 2,3 δεν είναι όλες οι μεταβλητές ακέραιες δημιουργούνται δυο νέοι κόμβοι από κάθε έναν από τους γονικούς. Η μεταβλητή που επιλέγεται σε κάθε έναν από τους γονικούς κόμβους είναι αυτή που βρίσκεται δεξιότερα στη λίστα και δεν έχει ακέραια τιμή. Οι τέσσερεις νέοι κόμβοι με τα όρια που θα εφαρμοστούν φαίνονται στο σχήμα 2 στην επόμενη σελίδα.



Σχήμα 2 **Οι τέσσερεις νέοι κόμβοι που σχηματίζονται με την εφαρμογή ορίων στους κόμβους 2 και 3.**

Κόμβος 4

Στον κόμβο 4 η μεταβλητή x_3 παίρνει την τιμή 0 και οι υπόλοιπες τιμές διατηρούν την τιμή που είχαν στον κόμβο 2 και άρα οι τιμές των παραμέτρων μεταβάλλονται και δίνονται στον πίνακα 16.

Πίνακας 16 **Οι τιμές των μεταβλητών και των παραμέτρων στην αρχή της επίλυσης του κόμβου 4.**

$x_2 = 9.75$	$I_{res} = 1$
$x_3 = 0 \leq 0$	$T_{res} = 0.25$
$r = 13 \leq 13$[5]	
$x_1 = x_4 = 0$	$z^* = 94.25$

Όπως και στον κόμβο 2 έτσι και εδώ υπάρχει διαθέσιμο απόθεμα, όμως εδώ υπάρχει και διαθέσιμη χρονική περίοδος, $T_{res} > 0$. Οπότε σε αυτόν τον κόμβο πρώτα θα πρέπει να μηδενιστεί, ικανοποιηθεί, ο 2ος περιορισμός. Η επίλυση γίνεται χρησιμοποιώντας και τις δυο μεθόδους επίλυσης για λόγους σύγκρισης.

Αρχική μεθοδολογία

Για να μηδενιστεί ο 2ος περιορισμός ο αλγόριθμος αναζητά την πρώτη μεταβλητή που μπορεί να αυξηθεί με βάση την κατάταξη των λόγων $\frac{R_i - R_j}{V_i - V_j}$ των μεταβλητών x, κινούμενος φυσικά από τα αριστερά προς τα δεξιά. Η μεταβλητή που επιλέγεται είναι η x_1 και θα πάρει

[5] Η ανισότητα αναφέρεται στο όριο που επιβλήθηκε στην r προηγουμένως και δείχνει ότι δεν παραβιάζεται.

την διαθέσιμη ποσότητα του 2ου περιορισμού, η οποία είναι 0.25. Στον πίνακα 17 φαίνονται οι νέες τιμές του προβλήματος.

Πίνακας 17 **Οι τιμές των μεταβλητών και των παραμέτρων αμέσως μόλις ικανοποιήθηκε ο 2ος περιορισμός.**

$x_1 = 0.25$	$I_{res} = 0.75$
$x_2 = 9.75$	$T_{res} = 0$
$r = 13 \leq 13$	
$x_3 = x_4 = 0$	$z^* = 95.25$

Όπως φαίνεται υπάρχει ακόμα διαθέσιμο απόθεμα οπότε τώρα θα ακολουθηθούν τα βήματα που έγιναν και στον κόμβο γονέα, δηλαδή τον κόμβο 2. Συνοπτικά θα αυξηθεί η πρώτη δεξιά μεταβλητή (δεξιότερα από την τελευταία που μεταβλήθηκε, δηλαδή τη x_1) και θα μειωθεί η πρώτη αριστερή μεταβλητή από αυτή που θα αυξηθεί. Άρα η x_2 θα αυξηθεί και η x_1 θα μειωθεί και η μεταβολή των τιμών τους θα είναι ίση με 0.25 καθώς μέχρι αυτό το νούμερο μπορεί να μειωθεί η x_1 ώστε να μην παραβιασθεί ο περιορισμός μη αρνητικότητας. Οι νέες τιμές της ενδιάμεσης λύσης του προβλήματος δίνονται στον πίνακα 18.

Νέα μεθοδολογία

Η διαδικασία εδώ είναι πολύ πιο απλή. Ο αλγόριθμος επιλέγει τη x_2 γιατί στο γονικό κόμβο 2 αυτή ήταν η μεταβλητή που μειώθηκε προκειμένου να αυξηθεί η τιμή της x_3. Οι τιμές των παραμέτρων και των μεταβλητών μετά από αυτές τις αλλαγές δίνονται στον πίνακα 18.

Πίνακας 18 **Οι τιμές των μεταβλητών και των παραμέτρων της ενδιάμεσης λύσης.**

$x_2 = 10$	$I_{res} = 0.5$
$r = 13 \leq 13$	$T_{res} = 0$
$x_1 = x_3 = x_4 = 0$	$z^* = 96$

Στη συνέχεια η διαδικασία είναι ίδια ανεξάρτητα από τη μεθοδολογία επίλυσης.

Επειδή ακόμα υπάρχει διαθέσιμο απόθεμα θα χρειαστεί και άλλη επανάληψη για την εύρεση της βέλτιστης λύσης του κόμβου 4. Η επανάληψη θα γίνει με την ίδια λογική που έγινε και προηγουμένως, αλλά εδώ η x_2 είναι αυτή που μειώνεται και η x_4 αυτή που αυξάνεται. Η τελική λύση δίνεται στον πίνακα 19 της επόμενης σελίδας.

Πίνακας 19 **Βέλτιστη λύση για τον κόμβο 4.**

$x_2 = 9.875$	$I_{res} = 0$
$x_4 = 0.125$	$T_{res} = 0$
$r = 13 \leq 13$	
$x_1 = x_3 = x_4 = 0$	z^* =96.375

Όπως φαίνεται η τιμή της αντικειμενικής αυξήθηκε σε σχέση με αυτή που είχε στην αρχή της επίλυσης του κόμβου 4, αλλά μειώθηκε σε σχέση με την τιμή που είχε στον κόμβο 1.

Κόμβος 5

Στον κόμβο 5 η μεταβλητή x_3 παίρνει την τιμή 1 και οι υπόλοιπες τιμές διατηρούν την τιμή που είχαν στον κόμβο 2 και άρα οι τιμές των παραμέτρων μεταβάλλονται και δίνονται στον πίνακα 20.

Πίνακας 20 **Οι τιμές των μεταβλητών και των παραμέτρων στην αρχή της επίλυσης του κόμβου 5.**

$x_2 = 9.75$	$I_{res} = -3$
$x_3 = 1 \geq 1$	$T_{res} = -0.75$
$r = 13 \leq 13$	
$x_1 = x_4 = 0$	$z^* = 103.25$

Η συνολική χρονική περίοδος εδώ έχει πάρει αρνητική τιμή, $T_{res} < 0$, το οποίο σημαίνει ότι χρησιμοποιούνται περισσότερες χρονικές περίοδοι από αυτές που υπάρχουν διαθέσιμες. Άρα θα πρέπει να μειωθεί κάποια μεταβλητή x η οποία έχει θετική τιμή και δεν έχει κάποιο όριο που θα την εμποδίζει να μειωθεί προκειμένου να γίνει 0 το T_{res}. Η μεταβλητή που ικανοποιεί αυτές τις προϋποθέσεις είναι η x_2 και θα μειωθεί η τιμή της κατά 0.75. Στον πίνακα 21 δίνονται οι τιμές των μεταβλητών και των παραμέτρων μετά την αλλαγή που μόλις έγινε.

Πίνακας 21 **Οι τιμές των μεταβλητών και των παραμέτρων αμέσως μόλις ικανοποιήθηκε ο 2ος περιορισμός.**

$x_2 = 9$	$I_{res} = -1.5$
$x_3 = 1 \geq 1$	$T_{res} = 0$
$r = 13 \leq 13$	
$x_1 = x_4 = 0$	$z^* = 98$

Η επίλυση συνεχίζεται και με τις δυο μεθόδους για σύγκριση της ταχύτητας επίλυσης.

Αρχική μεθοδολογία

Στη συνέχεια ελέγχεται ο $1^{ος}$ περιορισμός και όπως φαίνεται έχει και αυτός αρνητική τιμή όπως είχε συμβεί και στον κόμβο 3. Άρα ο αλγόριθμος θα αναζητήσει μια μεταβλητή βάσει των λόγων $\frac{R_i - R_j}{V_i - V_j}$ για τις μεταβλητές x που έχουν θετική τιμή και άρα μπορούν να μειωθούν και για την r χρησιμοποιείται ο λόγος $C = 2$. Η μεταβλητή που επιλέγεται είναι η r γιατί έχει τον μικρότερο λόγο. Επειδή η r δεν εμφανίζεται στον 2^o περιορισμό δε χρειάζεται να αυξηθεί κάποια άλλη τιμή και έτσι η r θα μειωθεί κατά 1.5 μονάδα και θα πάρει την τιμή 11.5, η οποία είναι μικρότερη του 13, όπως επιβάλλει το όριο. Η βέλτιστη λύση του κόμβου 5 δίνεται στον πίνακα 22.

Νέα μεθοδολογία

Με τη νέα μεθοδολογία επίλυσης επιλέγεται η r γιατί είναι η πρώτη μεταβλητή αριστερά της x_3 και η οποία είναι μεγαλύτερη από το κάτω όριο της και άρα μπορεί να μειωθεί. Τα αποτελέσματα αυτής της επανάληψης δίνονται στον πίνακα 22.

Πίνακας 22 **Βέλτιστη λύση για τον κόμβο 5.**

$x_2 = 9$	$I_{res} = 0$
$x_3 = 1 \geq 1$	$T_{res} = 0$
$r = 11.5 \leq 13$	
$x_1 = x_4 = 0$	$z^* = 95$

Όπως παρατηρείται η τιμή της αντικειμενικής μειώθηκε σε σχέση με αυτή που είχε αμέσως μετά την εφαρμογή του ορίου στη x_3, το οποίο ήταν αναμενόμενο αφού αρχικά είχε δαπανηθεί περισσότερο απόθεμα από το διαθέσιμο. Επιπλέον, η τελική τιμή του κόμβου είναι μικρότερη από αυτή του κόμβου 1, το οποίο είναι επίσης λογικό και αναμενόμενο.

Είναι προφανές ότι η νέα μεθοδολογία παράγει γρηγορότερα αποτελέσματα από την αρχική χρησιμοποιώντας τη λίστα. Για αυτό το λόγο για τους υπόλοιπους κόμβους θα αναλύεται μόνο η νέα μεθοδολογία, η οποία χρησιμοποιείται και στον κώδικα.

Κόμβος 6

Στον κόμβο 6, που προήλθε από τον 3, η x_2 παίρνει την τιμή 9 και οι υπόλοιπες μεταβλητές διατηρούν τις τιμές τους, οι οποίες δίνονται στον πίνακα 23 μαζί με τις νέες τιμές των παραμέτρων.

Πίνακας 23 **Οι τιμές των μεταβλητών και των παραμέτρων στην αρχή της επίλυσης του κόμβου 6.**

$x_1 = 0.5$	$I_{res} = 1$
$x_2 = 9 \leq 9$	$T_{res} = 0.5$
$r = 14 \geq 14$	
$x_3 = x_4 = 0$	$z^* = 93$

Όπως φαίνεται υπάρχουν και απόθεμα και χρονικοί περίοδοι διαθέσιμοι, ακριβώς όπως στον κόμβο 4 οπότε θα ακολουθηθεί η ίδια ακριβώς μεθοδολογία. Στην πρώτη επανάληψη θα πρέπει το T_{res}να πάρει την τιμή 0 και σε επόμενη ή επόμενες επαναλήψεις θα μηδενιστεί και το I_{res} ώστε να προκύψει η βέλτιστη λύση του κόμβου 6. Στον πίνακα 24 δίνονται οι τιμές των μεταβλητών και των παραμέτρων για μηδενισμό του T_{res}.

Πίνακας 24 **Οι τιμές των μεταβλητών και των παραμέτρων αμέσως μόλις ικανοποιήθηκε ο 2ος περιορισμός.**

$x_1 = 1$	$I_{res} = 0.5$
$x_2 = 9 \leq 9$	$T_{res} = 0$
$r = 14 \geq 14$	
$x_3 = x_4 = 0$	$z^* = 95$

Επειδή το I_{res} έχει θετική τιμή χρειάζεται και άλλη επανάληψη, στην οποία αυξάνεται η τιμή της μεταβλητής r για την εύρεση της βέλτιστης λύσης του κόμβου 6, πίνακας 25.

Πίνακας 25 **Βέλτιστη λύση για τον κόμβο 6.**

$x_1 = 1$	$I_{res} = 0$
$x_2 = 9 \leq 9$	$T_{res} = 0$
$r = 14.5 \geq 14$	
$x_1 = x_4 = 0$	$z^* = 96$

Κόμβος 7

Στον κόμβο 7 η μεταβλητή x_2 παίρνει την τιμή 10 και οι τιμές που παίρνουν οι παράμετροι έχουν αρνητικές τιμές όπως στον κόμβο 5, όπως φαίνεται στον πίνακα 26.

Πίνακας 26 **Οι τιμές των μεταβλητών και των παραμέτρων στην αρχή της επίλυσης του κόμβου 7.**

$x_1 = 0.5$	$I_{res} = -1$
$x_2 = 10 \geq 10$	$T_{res} = -0.5$
$r = 14 \geq 14$	
$x_1 = x_4 = 0$	$z^* = 100$

Για την εύρεση της βέλτιστης λύσης του κόμβου 7 χρησιμοποιείται η ίδια μεθοδολογία με αυτή που χρησιμοποιήθηκε στον κόμβο 5. Αρχικά θα πρέπει ο $2^{ος}$ περιορισμός να μηδενιστεί και αυτό θα γίνει με μείωση της τιμής της x_1 η οποία θα γίνει 0. Στον πίνακα 27 φαίνονται οι νέες τιμές που προέκυψαν από αυτήν την αλλαγή.

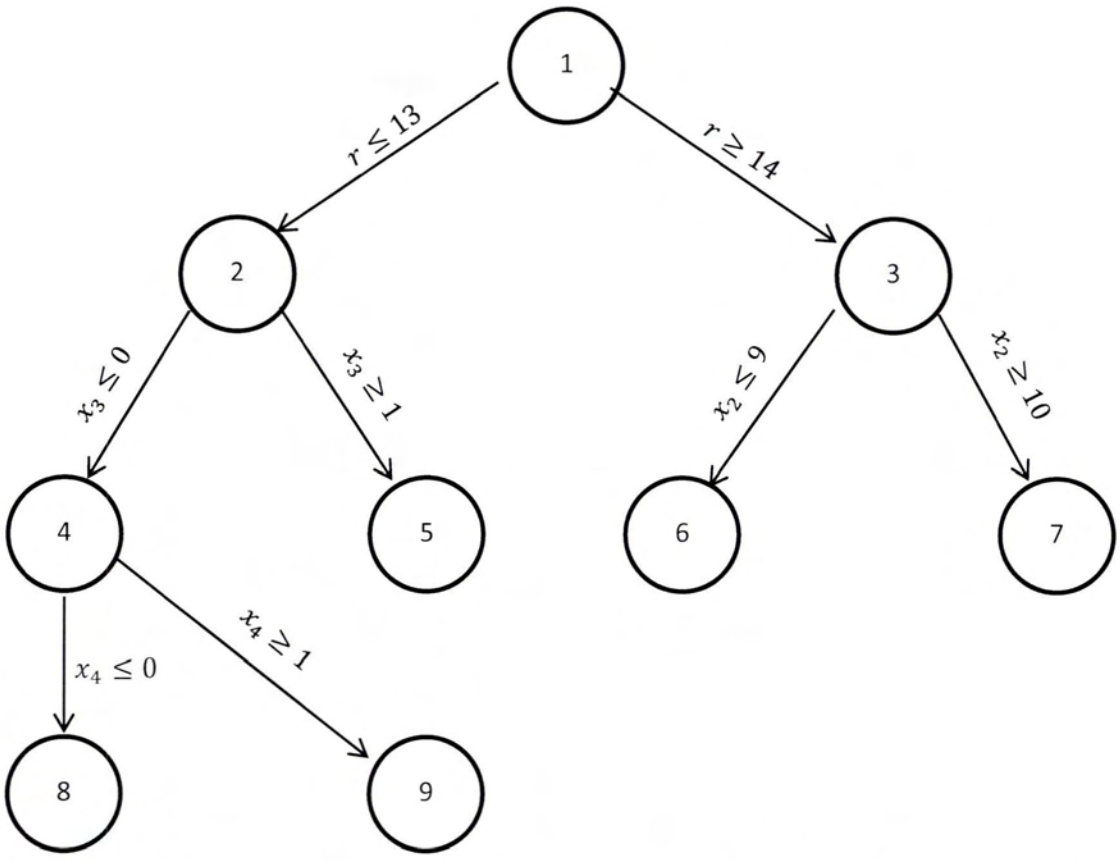
Πίνακας 27 **Οι τιμές των μεταβλητών και των παραμέτρων αμέσως μόλις ικανοποιήθηκε ο $2^{ος}$ περιορισμός.**

$x_2 = 10 \geq 10$	$I_{res} = -0.5$
$r = 14 \geq 14$	$T_{res} = 0$
$x_1 = x_3 = x_4 = 0$	$z^* = 98$

Επειδή το διαθέσιμο απόθεμα έχει αρνητική τιμή θα χρειαστεί να γίνει και άλλη επανάληψη κατά την οποία θα πρέπει να μειωθεί είτε η μεταβλητή x_2 είτε η r. Παρόλα αυτά κάτι τέτοιο είναι αδύνατον γιατί και οι δύο μεταβλητές έχουν πάρει την ελάχιστη δυνατή τιμή τους οπότε ο κόμβος 7 δεν έχει εφικτή λύση και άρα δεν θα γίνει κόμβος – γονέας στη συνέχεια της επίλυσης του προβλήματος.

Γονικοί κόμβοι 4,5,6

Επειδή και στους τρεις κόμβους 4, 5 και 6 δεν είναι όλες οι μεταβλητές ακέραιες δημιουργούνται δυο νέοι κόμβοι από κάθε έναν από τους γονικούς. Η μεταβλητή που επιλέγεται σε κάθε έναν από τους γονικούς κόμβους είναι αυτή που βρίσκεται δεξιότερα στη λίστα και δεν έχει ακέραια τιμή. Αρχικά δημιουργούνται δυο νέοι κόμβοι από τον κόμβο 4 όπου η αντικειμενική έχει τη μεγαλύτερη τιμή σε σχέση με τους κόμβους 5 και 6. Η εξέλιξη του δέντρου branch and bound, μέχρι αυτό το σημείο δίνεται στο σχήμα 3 της επόμενης σελίδας.



Σχήμα 3 **Οι δυο νέοι κόμβοι που σχηματίζονται με την εφαρμογή ορίων στον κόμβο 4, στον οποίο η αντικειμενική έχει τη μεγαλύτερη τιμή ανάμεσα στους ενεργούς κόμβου 5 και 6.**

Κόμβος 8

Στον κόμβο 8 η x_4 παίρνει την τιμή 0 οπότε μεταβάλλονται και οι τιμές των παραμέτρων όπως φαίνεται στον πίνακα 28.

Πίνακας 28**Οι τιμές των μεταβλητών και των παραμέτρων στην αρχή της επίλυσης του κόμβου 8.**

$x_2 = 9.875$	$I_{res} = 0.75$
$x_4 = 0 \leq 0$	$T_{res} = 0.125$
$x_3 = 0 \leq 0$	
$r = 13 \leq 13$	
$x_1 = 0$	$z^* = 95.125$

Στο σημείο αυτό το πρόβλημα έχει ακριβώς την ίδια μορφή με τον κόμβο 4, θετικά T_{res}και I_{res}, οπότε θα χρησιμοποιηθεί η ίδια μεθοδολογία. Έτσι επιλέγεται μεταβλητή x_2 για να αυξηθεί γιατί στο γονικό κόμβο ήταν αυτή που μειώθηκε προκείμενου να μπορέσει να αυξηθεί η x_4. Από αυτή τη μεταβολή προκύπτουν τα αποτελέσματα του πίνακα 29 στην επόμενη σελίδα.

Πίνακας 29 **Βέλτιστη λύση για τον κόμβο 8.**

$x_2 = 10$	$I_{res} = 0.5$
$x_1 = 0$	$T_{res} = 0$
$x_4 = 0 \leq 0$	
$x_3 = 0 \leq 0$	
$r = 13 \leq 13$	$z^* = 96$

Παρότι υπάρχει ακόμα διαθέσιμο απόθεμα δεν υπάρχει κάποια «κίνηση» που να επιτρέπεται για αύξηση της τιμής της αντικειμενικής. Βάσει του αλγορίθμου θα έπρεπε η πρώτη μεταβλητή δεξιά της x_2 να αυξηθεί αλλά κάτι τέτοιο είναι αδύνατον γιατί και οι τρεις δεξιές μεταβλητές έχουν λάβει τη μέγιστη τιμή τους, οπότε η λύση του πίνακα 29 θα αποτελεί τη βέλτιστη για τον κόμβο 8. Η λύση αυτή είναι ακέραια οπότε από τον κόμβο 8 δεν μπορούν να δημιουργηθούν νέοι κόμβοι.

Κόμβος 9

Στον κόμβο 9 η μεταβλητή x_4 παίρνει την τιμή 1 και στον πίνακα 30 φαίνονται τα αποτελέσματα της μεταβολής αυτής.

Πίνακας 30 **Οι τιμές των μεταβλητών και των παραμέτρων στην αρχή της επίλυσης του κόμβου 9.**

$x_2 = 9.875$	$I_{res} = -5.25$
$x_4 = 1 \geq 1$	$T_{res} = -0.875$
$x_3 = 0 \leq 0$	
$r = 13 \leq 13$	
$x_1 = 0$	$z^* = 105.125$

Η μορφή των αποτελεσμάτων είναι ίδια με αυτή του κόμβου 5 οπότε θα ακολουθηθεί η ίδια διαδικασία για την εύρεση της βέλτιστης λύσης του κόμβου 9. Συγκεκριμένα, αρχικά θα μειωθεί η τιμή της μεταβλητής x_2 κατά 0.875 για να μηδενιστεί το T_{res}. Οι τιμές που προκύπτουν μετά από αυτή την αλλαγή δίνονται στον πίνακα 31.

Πίνακας 31 **Οι τιμές των μεταβλητών και των παραμέτρων αμέσως μόλις ικανοποιήθηκε ο 2ος περιορισμός.**

$x_2 = 9$	$I_{res} = -3.5$
$x_4 = 1 \geq 1$	$T_{res} = 0$
$x_3 = 0 \leq 0$	
$r = 13 \leq 13$	
$x_1 = 0$	$z^* = 99$

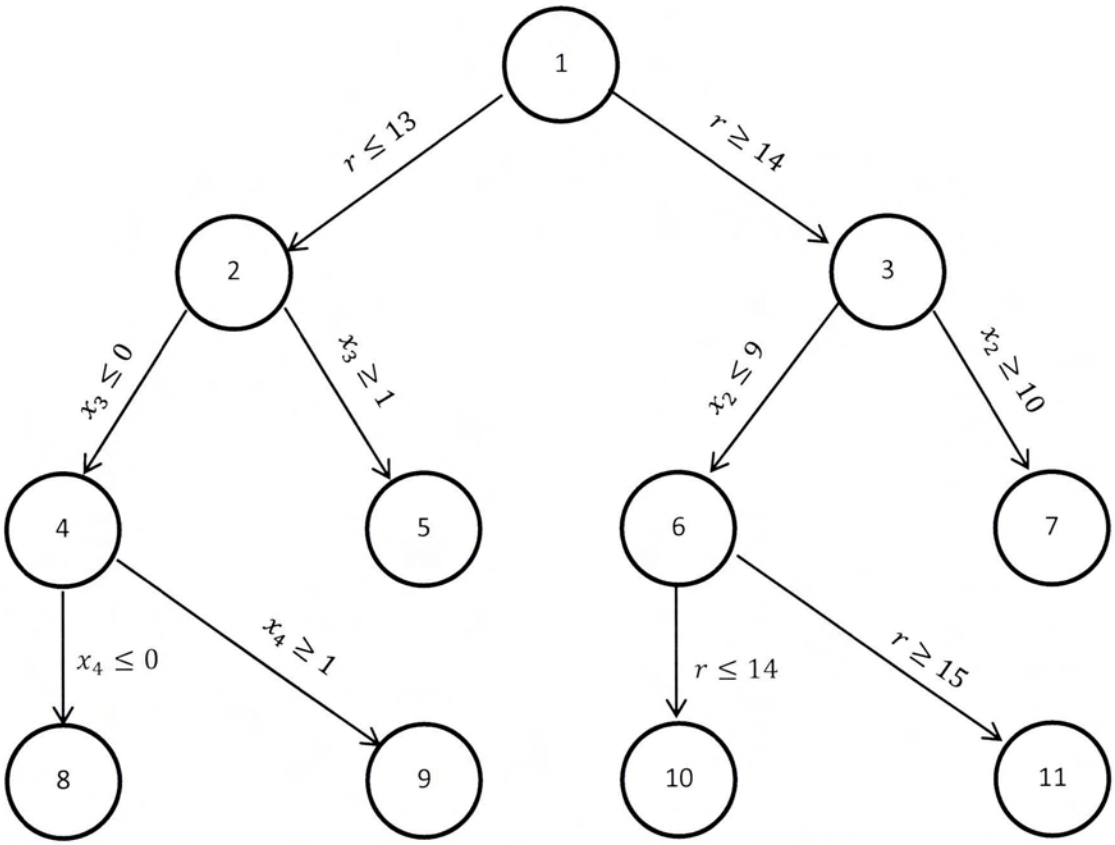
Στη συνέχεια ακριβώς όπως στον κόμβο 5 θα πρέπει να μηδενιστεί το I_{res} και αυτό θα γίνει με μείωση της τιμής της μεταβλητής r και η βέλτιστη λύση του κόμβου 9 φαίνεται στον πίνακα 32.

Πίνακας 32 **Βέλτιστη λύση για τον κόμβο 9.**

$x_2 = 9$	$I_{res} = 0$
$x_4 = 1 \geq 1$	$T_{res} = 0$
$x_3 = 0 \leq 0$	
$r = 9.5 \leq 13$	
$x_1 = 0$	$z^* = 92$

Γονικοί κόμβοι 5, 6, 8, 9

Σε αυτό το σημείο ελέγχονται οι τιμές της αντικειμενικής των κόμβων 5, 6, 8 και 9. Από τη σύγκριση φαίνεται ότι ο κόμβος 6 έχει τη μεγαλύτερη τιμή οπότε από αυτόν θα δημιουργηθούν 2 νέοι κόμβοι, οι 10 και 11, όπως φαίνεται στο σχήμα 4 παρακάτω.



Σχήμα 4 **Οι δυο νέοι κόμβοι που σχηματίζονται με την εφαρμογή ορίων στον κόμβο 6.**

Θα μπορούσε να είχε επιλεγεί και ο κόμβος 8, επειδή η αντικειμενική έχει την ίδια τιμή με αυτή του κόμβου 6, αλλά σε αυτόν δεν υπάρχουν μη ακέραιες μεταβλητές.

Κόμβος 10

Στους κόμβους 10 και 11 πάλι θέτονται όρια στη μεταβλητή r. Συγκεκριμένα, στον 10 η r θα πάρει την τιμή 14 και στον πίνακα 33 φαίνονται οι τιμές των μεταβλητών και των παραμέτρων.

Πίνακας 33 **Οι τιμές των μεταβλητών και των παραμέτρων στην αρχή της επίλυσης του κόμβου 10.**

$x_1 = 1$	$I_{res} = 0.5$
$x_2 = 9 \leq 9$	$T_{res} = 0$
$14 \leq r = 14 \geq 14$	
$x_3 = x_4 = 0$	$z^* = 95$

Το διαθέσιμο απόθεμα έχει θετική τιμή και η μορφή που έχει αυτή τη στιγμή το πρόβλημα είναι παρόμοια με προηγούμενους κόμβους, όπως ο κόμβος 2 αμέσως μετά την εφαρμογή των ορίων, οπότε θα ακολουθηθεί η ίδια διαδικασία. Αυτό σημαίνει ότι θα αυξηθεί η τιμή της x_3 και αντίστοιχα θα μειωθεί η τιμή της x_1. Οι νέες τιμές, οι οποίες αποτελούν τη βέλτιστη λύση του κόμβου 10, δίνονται στον πίνακα 34.

Πίνακας 34 **Η βέλτιστη λύση για τον κόμβο 10.**

$x_1 = 0.833$	$I_{res} = 0$
$x_2 = 9 \leq 9$	$T_{res} = 0$
$x_3 = 0.167$	
$14 \leq r = 14 \geq 14$	
$x_4 = 0$	$z^* = 95.83$

Κόμβος 11

Στον κόμβο 11 η r θα πάρει την τιμή 15 οπότε θα μεταβληθούν οι τιμές των παραμέτρων με τα αποτελέσματα των μεταβολών αυτών να φαίνονται στον πίνακα 35.

Πίνακας 35 **Οι τιμές των μεταβλητών και των παραμέτρων στην αρχή της επίλυσης του κόμβου 11.**

$x_1 = 1$	$I_{res} = -0.5$
$x_2 = 9 \leq 9$	$T_{res} = 0$
$r = 15 \geq 15$	
$x_3 = x_4 = 0$	$z^* = 97$

Επειδή η μορφή του προβλήματος είναι ίδια με αυτή του κόμβου 3, αμέσως μετά την εφαρμογή των ορίων, θα ακολουθηθεί η ίδια μεθοδολογία οπότε θα αυξηθεί η τιμή της x_1 και θα μειωθεί η τιμή της x_2. Στον πίνακα 36 δίνεται η βέλτιστη λύση του κόμβου 11.

Πίνακας 36 **Η βέλτιστη λύση για τον κόμβο 11.**

$x_1 = 1.5$	$I_{res} = 0$
$x_2 = 8.5 \leq 9$	$T_{res} = 0$
$r = 15 \geq 15$	
$x_3 = x_4 = 0$	$z^* = 95.5$

Βέλτιστη λύση του προβλήματος

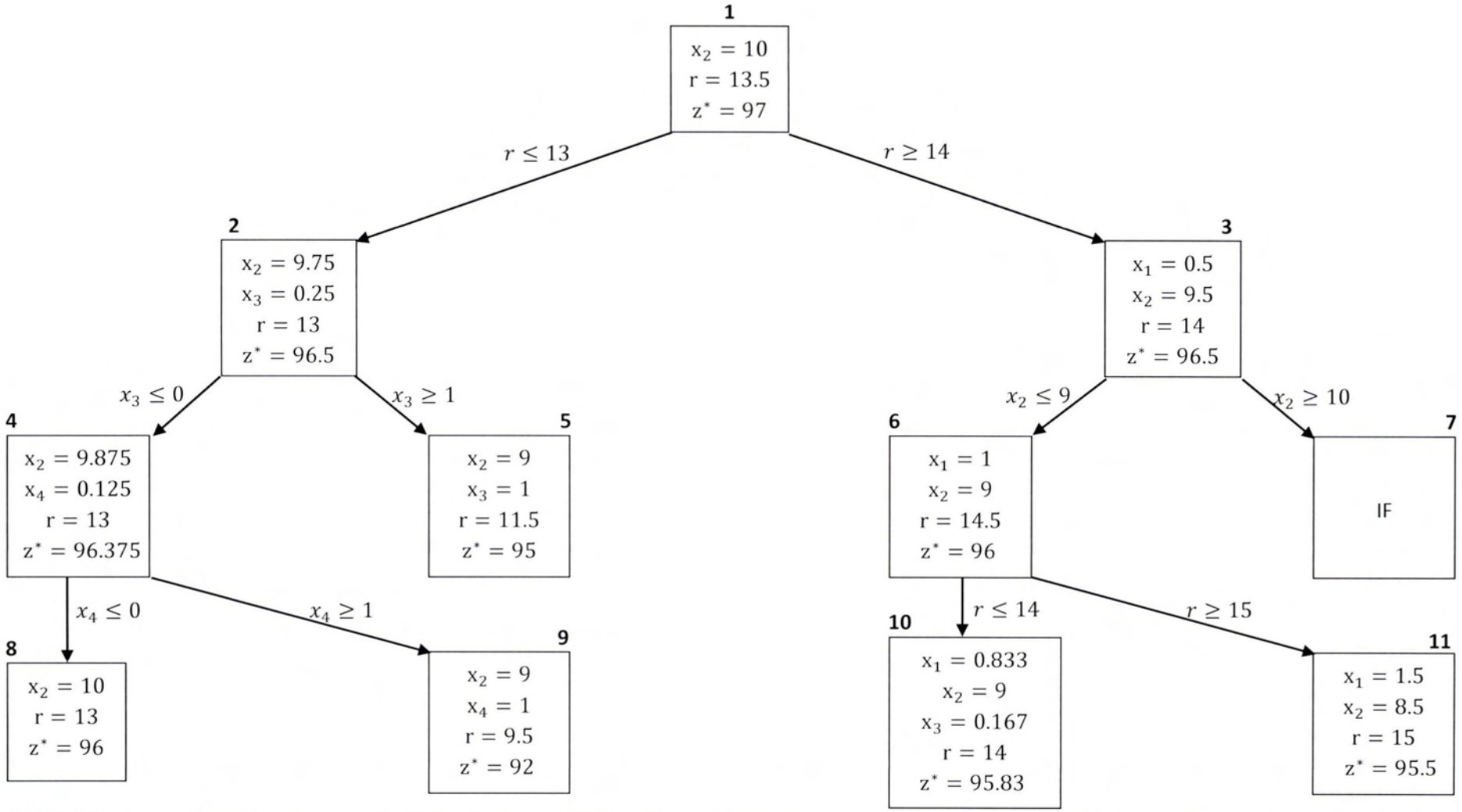
Στη συνέχεια ο αλγόριθμος ελέγχει τις τιμές της αντικειμενικής για τους κόμβους 5, 8, 9, 10 και 11 και επιλέγει αυτόν με την μεγαλύτερη τιμή για να συνεχίσει. Συνεπώς επιλέγεται ο κόμβος 8, ο οποίος έχει τη μεγαλύτερη τιμή, όπου όμως όλες οι μεταβλητές έχουν ακέραιες τιμές. Αυτό σημαίνει ότι ο κόμβος 8 αποτελεί τη βέλτιστη λύση του προβλήματος, αφού όσο αναπτύσσεται το δέντρο οι τιμές της αντικειμενικής συνεχώς θα μειώνονται οπότε δεν θα είχε νόημα να συνεχίσει σε κάποιον άλλον κόμβο, με μη ακέραιες τιμές μεταβλητών, αναζητώντας μεγαλύτερη τιμή. Η λύση του κόμβου 8 ξαναδίνεται στον πίνακα 37.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ ΒΙΒΛΙΟΘΗΚΗ

Πίνακας 37 **Βέλτιστη λύση του προβλήματος.**

$x_2 = 10$	$I_{res} = 0.5$
$x_1 = 0$	$T_{res} = 0$
$x_4 = 0 \leq 0$	
$x_3 = 0 \leq 0$	
$r = 13 \leq 13$	$z^* = 96$

Στο σχήμα 5 της επόμενης σελίδας φαίνεται συνολικά το δέντρο branch and bound με όλους τους περιορισμούς και όλες τις τιμές των μη μηδενικών μεταβλητών ανά κόμβο.



Σχήμα 5 **Ολόκληρο το δέντρο branch and bound για την εύρεση της βέλτιστης λύσης του παραδείγματος, η οποία βρίσκεται στον κόμβο 8. Σε κάθε κόμβο δίνονται οι τιμές των μη μηδενικών μεταβλητών και της αντικειμενικής.**

5. Περιγραφή κώδικα

Για την ανάπτυξη του αλγορίθμου όλες οι παραπάνω ιδιότητες γράφτηκαν σε γλώσσα προγραμματισμού C και χρησιμοποιώντας το πρόγραμμα Microsoft Visual Studio. Ταυτόχρονα για να συγκριθεί η ταχύτητα και η ορθότητα των αποτελεσμάτων χρησιμοποιήθηκε και το λογισμικό πακέτο βελτιστοποίησης CPLEX.

Οι συναρτήσεις που χρησιμοποιήθηκαν για την υλοποίηση του αλγορίθμου αναφέρονται παρακάτω και ακολουθεί μία σύντομη περιγραφή στις επόμενες σελίδες για κάθε μια από αυτές:

- Insert_node_to_tail
- Insert_node_in_place
- InsertTreeNode
- DeleteTreeNode
- Make_list
- Insert_data
- GetPrevDecrX
- GetPrevIncrX
- GetNextIncrX
- Objective_value
- GoRight
- GoLeft
- Print_results
- Free_list
- UndoLeft
- FinishRight
- Kozalg

Insert_node_to_tail

Κατά τη δημιουργία της λίστας μεταβλητών, όταν βρίσκεται η επόμενη x_i που θα εισαχθεί, καλείται η συνάρτηση *insert_node_to_tail* η οποία βάζει τη μεταβλητή στο τέλος της ήδη υπάρχουσας λίστας.

Insert_node_in_place

Μετά την εισαγωγή όλων των μεταβλητών x_i στη λίστα, καλείται η συνάρτηση *insert_node_in_place* η οποία συγκρίνει τον λόγο της r με τους λόγους των υπόλοιπων μεταβλητών και την εισάγει στην κατάλληλη θέση στη λίστα, έτσι ώστε οι τιμές τους να ακολουθούν φθίνουσα πορεία.

InsertTreeNode

Μετά την εύρεση λύσης κάθε κόμβου του δέντρου, δημιουργείται ένα *TNode* το οποίο περιέχει πληροφορίες για τη λύση αυτή, πιο συγκεκριμένα περιέχει τα υπόλοιπα των δύο περιορισμών, την τιμής της αντικειμενικής και των μεταβλητών, τα άνω και κάτω όρια καθώς και τον αριθμό του κόμβου του δέντρου στον οποίο αντιστοιχεί. Για να συνεχίζεται η ανάπτυξη του δέντρου από τον κόμβο με τη μεγαλύτερη τιμή αντικειμενικής, οι λύσεις μπαίνουν σε μία λίστα, παρόμοια με αυτή των μεταβλητών. Η συνάρτηση *InsertTreeNode* εισάγει την κάθε λύση στη σωστή θέση της λίστας αυτής, ώστε να βρίσκονται οι αντικειμενικές των κόμβων σε φθίνουσα σειρά.

DeleteTreeNode

Αφότου αναπτυχθεί ένας κόμβος γονέας, παράγοντας τους δύο αντίστοιχους με άνω και κάτω όριο σε μία μεταβλητή, πρέπει το αντίστοιχο *TNode* που περιείχε τις πληροφορίες του να διαγραφεί από τη λίστα. Τη δουλειά αυτή αναλαμβάνει η *DeleteTreeNode*.

Make_list

Για να δημιουργηθεί η λίστα των μεταβλητών, η *make_list* συγκρίνει αρχικά σε κάθε επανάληψη τους λόγους των x_i. Αφού βρει τον μεγαλύτερο, εισάγει την αντίστοιχη μεταβλητή στη λίστα καλώντας την *insert_node_to_tail*. Αφότου τελειώσει η διαδικασία αυτή, καλεί την *insert_node_in_place* ώστε να εισαχθεί και η r.

Insert_data

Στη διαδικασία ελέγχου του αλγορίθμου δοκιμάστηκαν διάφορα προβλήματα, τα νούμερα των οποίων βγήκαν από τη γεννήτρια που εμπεριέχεται στην *insert_data*. Από αυτήν παράγονται όλες οι παράμετροι του προβλήματος (R_i, V_i, I, T) μόνο όταν είναι σε σχόλια η εντολή *define user_input.*

GetPrevDecrX

Αναζητά και επιστρέφει, όταν καλείται, την πρώτη μεταβλητή x_i, αριστερά μιας συγκεκριμένης x_j, $(j \neq i)$ που μπορεί να μειωθεί.

GetPrevIncrX

Αναζητά και επιστρέφει, όταν καλείται, την πρώτη μεταβλητή x_i, αριστερά μιας συγκεκριμένης $x_j, (j \neq i)$ που μπορεί να αυξηθεί.

GetNextIncrX

Αναζητά και επιστρέφει, όταν καλείται, την πρώτη μεταβλητή x_i, δεξιά μιας συγκεκριμένης $x_j, (j \neq i)$ που μπορεί να αυξηθεί.

Objective_value

Μετά την εύρεση της ακέραιας λύσης του προβλήματος καλείται η *objective_value* ώστε να βρεθεί η τιμή της αντικειμενικής.

GoRight

Στις περιπτώσεις δημιουργίας κόμβου με άνω όριο, τα υπόλοιπα των δεξιών μελών των δύο περιορισμών αλλάζουν και είναι πλέον μεγαλύτερα του μηδέν. Η *GoRight* καλείται ώστε να μοιραστεί το υπόλοιπο αυτό με τον βέλτιστο τρόπο σε μεταβλητές δεξιότερα της τελευταίας που πήρε τιμή (στη συνάρτηση είναι η *Next*). Οι πιθανές περιπτώσεις είναι:

- Η επόμενη μεταβλητή είναι η r, άρα η αύξησή της γίνεται ανεξάρτητα, χωρίς να χρειάζεται να βρεθεί μεταβλητή να μειωθεί.
- Το υπόλοιπο του δεύτερου περιορισμού (T_{res}) είναι θετικό, άρα η αύξηση θα γίνει στη μεταβλητή Next χωρίς την αντίστοιχη μείωση μιας άλλης και θα ξαναυπολογιστεί το υπόλοιπο του πρώτου περιορισμού (I_{res}).
- Μόνο το υπόλοιπο του πρώτου περιορισμού (I_{res}) είναι θετικό, άρα αναζητείται η πρώτη μεταβλητή αριστερά της Next που μπορεί να μειωθεί.

Οι επαναλήψεις τελειώνουν είτε όταν δεν υπάρχουν άλλες μεταβλητές που να μπορούν να αυξηθούν, είτε όταν το υπόλοιπο του πρώτου περιορισμού γίνει μηδέν.

GoLeft

Αντίστοιχα με την *GoRight* όταν δημιουργείται ο κόμβος με το κάτω όριο, το δεξί μέλος του πρώτου περιορισμού θα είναι αρνητικό. Σε αυτή τη περίπτωση η *GoLeft* διορθώνει το υπόλοιπο ώστε να γίνει μηδέν, ξεκινώντας από τη μεταβλητή που βρίσκεται πιο αριστερά της τελευταίας μεταβλητής που πήρε τιμή (στη συνάρτηση είναι η *Next*). Οι πιθανές περιπτώσεις είναι οι εξής:

- Η πρώτη μεταβλητή που μπορεί να μειωθεί είναι η r, οπότε και η μείωσή της γίνεται ανεξάρτητα των υπολοίπων.

- Το υπόλοιπο του δεύτερου περιορισμού (T_{res}) είναι θετικό, άρα μειώνεται η μεταβλητή *Next* χωρίς την ταυτόχρονη αύξηση κάποιας άλλης x_i.
- Μόνο το υπόλοιπο του πρώτου περιορισμού (I_{res}) είναι αρνητικό, οπότε αναζητείται μία μεταβλητή αριστερά της *Next* που να μπορεί να αυξηθεί. Αν η μεταβλητή αυτή υπάρχει τότε γίνεται η ταυτόχρονη μεταβολή των τιμών τους. Σε περίπτωση που δεν υπάρχει όμως, τότε αναζητείται η πρώτη που μπορεί να μειωθεί ανεξάρτητα. Αν πάλι δεν υπάρχει τέτοια μεταβλητή το πρόβλημα είναι μη εφικτό (κάτι που ορίζεται από τη τιμή του *fracvalue* που γίνεται -1).

Όπως στην *GoRight*, οι επαναλήψεις τελειώνουν είτε όταν δεν υπάρχουν διαθέσιμες μεταβλητές προς μείωση, είτε όταν το υπόλοιπο του πρώτου περιορισμού γίνει μηδέν.

Print_results

Τυπώνει επιλεκτικά αποτελέσματα τόσο στην οθόνη, όσο και σε αρχείο.

Free_list

Καλείται για να ελευθερώσει τη μνήμη που είχε δεσμευτεί από τη λίστα των μεταβλητών μετά την εύρεση της βέλτιστης λύσης.

UndoLeft

Αφότου έχει βρεθεί η λύση σε έναν κόμβο, καλείται η *UndoLeft* ώστε να δημιουργηθεί ο κόμβος με το άνω όριο στη μη - ακέραια μεταβλητή που βρίσκεται πιο δεξιά στη λίστα. Η συνάρτηση ακολουθεί τα εξής βήματα:

- Εφαρμόζει το άνω όριο και μοιράζει το υπόλοιπο που προκύπτει από τη νέα τιμή της μεταβλητής.
- Αν πρόκειται για την μεταβλητή r, τότε ανανεώνει το υπόλοιπο του πρώτου περιορισμού και τη τιμή της αντικειμενικής.
- Αν η αλλαγή έγινε σε μεταβλητή x_i, φροντίζει να μοιράσει το θετικό υπόλοιπο του πρώτου περιορισμού σε κάποια μεταβλητή αριστερά της x_i που μπορεί να αυξηθεί. Στη συνέχεια αλλάζει τα υπόλοιπα T_{res} και I_{res}.

FinishRight

Ομοίως με την *UndoLeft,* η *FinishRight* καλείται για να δημιουργηθεί ο κόμβος με το κάτω όριο στη μη - ακέραια μεταβλητή που βρίσκεται πιο δεξιά στη λίστα. Η συνάρτηση ακολουθεί τα παρακάτω βήματα:

- Εφαρμόζει το κάτω όριο στη μεταβλητή.

- Αν η μεταβλητή ήταν η r, τότε αφότου αλλάξει την τιμή της, αλλάζει το υπόλοιπο του πρώτου περιορισμού και την τιμή της αντικειμενικής.
- Αν όμως η αλλαγή έγινε σε x_i, τότε έχοντας εφαρμόσει αυτό το όριο, η συνάρτηση αναζητεί μεταβλητές αριστερά της x_i, που μπορούν να μειωθούν προσπαθώντας να αναιρέσει την τελευταία κίνηση.
    - Αν δεν υπάρχουν τέτοιες, τότε το πρόβλημα δεν είναι εφικτό και για αυτό αλλάζει το *fracvalue* και γίνεται -1.
    - Στην αντίθετη όμως περίπτωση, μειώνει τη μεταβλητή αυτή και στη συνέχεια μέσω της *GoLeft* αναζητεί περεταίρω τρόπους να επαναφέρει το υπόλοιπο του πρώτου και του δεύτερου περιορισμού στο μηδέν.

Kozalg

Μέσω της *Kozalg* καλούνται όλες οι παραπάνω συναρτήσεις ώστε να βρεθεί η βέλτιστη λύση του προβλήματος. Η ανάπτυξη του δέντρου σταματάει όταν δεν υπάρχει κόμβος που μπορεί να αναπτυχθεί περαιτέρω ή όταν η λύση του κόμβου που βρίσκεται στη πρώτη θέση της λίστας και έχει την μεγαλύτερη τιμή αντικειμενικής είναι ακέραια.

6. Υπολογιστικά πειράματα

Στα πλαίσια της εργασίας επιλύθηκαν διάφορα προβλήματα με τη χρήση της γεννήτριας που περιέχεται στη συνάρτηση *insert_data* για τη διαφοροποίηση των παραμέτρων των προβλημάτων. Όλοι οι αριθμοί που χρησιμοποιήθηκαν παρήχθησαν σε αυτή τη συνάρτηση βάσει των τύπων που αναφέρονται στη μορφοποίηση παραπάνω.

Συγκεκριμένα επιλύθηκαν 50 παραδείγματα - προβλήματα για πέντε διαφορετικούς αριθμούς x_i μεταβλητών N, 1000, 5000, 10000, 15000 και 20000 μεταβλητές. Για κάθε αριθμό μεταβλητών N έγιναν 10 τυχαία παραδείγματα. Για τα παραδείγματα αυτά δίνονται οι χρόνοι επίλυσης τόσο με τη CPLEX όσο και με το συγκεκριμένο αλγόριθμο, ενώ οι λύσεις είναι ίδιες. Επίσης, δίνονται οι τιμές της αντικειμενικής της γραμμική χαλάρωση και της ακέραιας λύσης μαζί με τους κόμβους που χρειάστηκαν για την εύρεσή της.

Τα αποτελέσματα από τα 50 παραδείγματα – προβλήματα δίνονται στους πίνακες 38 – 42. Στη συνέχεια ακολουθεί ο πίνακας 43 με τα συνοπτικά αποτελέσματα προκειμένου να γίνει πιο εύκολη η σύγκριση αυτών.

Με χρήση του Microsoft Visual Studio οι χρόνοι υπολογίζονται με ακρίβεια 3 δεκαδικών ψηφίων, ενώ οι τιμές των αντικειμενικών με ακρίβεια 6 ψηφίων. Στις αποκλίσεις των χρόνων το θετικό πρόσημο σημαίνει ότι ο αλγόριθμος είναι γρηγορότερος από τη CPLEX, ενώ αρνητικό πρόσημο δηλώνει ότι η CPLEX είναι γρηγορότερη. Στις αποκλίσεις των τιμών των αντικειμενικών πάντα το πρόσημο θα είναι θετικό γιατί η γραμμική χαλάρωση θα δίνει πάντα μεγαλύτερη τιμή από την ακέραια λύση.

Ο αριθμός των κόμβων του αλγορίθμου αναφέρεται στο σύνολο των κόμβων που δημιουργήθηκαν προκειμένου να επιλυθεί κάθε ένα από τα 50 παραδείγματα και δεν αναφέρονται στον κόμβο που περιέχει τη βέλτιστη λύση. Η μέση τιμή των κόμβων έχει στρογγυλοποιηθεί προς τον επόμενο ακέραιο σε περίπτωση που η τιμή ήταν δεκαδική.

Για N = 1000

Πίνακας 38 **Αποτελέσματα των 10 υπολογιστικών πειραμάτων για 1000 x_i μεταβλητές.**

α/α	Χρόνος (sec)			Τιμές αντικειμενικών			Κόμ-βοι
	CPLEX	Αλγό-ριθμος	Από-κλιση	Γραμμική χαλάρωση	Τελική ακέραια λύση	Απόκλιση	
1	0.248	0.017	0.231	361727.252126	361725.404836	1.847290	49
2	0.147	0.016	0.131	47148.298900	47141.117871	7.181029	15
3	0.111	0.019	0.092	71774.305738	71772.642553	1.663185	77
4	0.146	0.013	0.133	1252.276319	1250.901896	1.374423	5
5	0.130	0.015	0.115	29442.151800	29439.883161	2.268639	25
6	0.136	0.017	0.119	95715.754804	95714.954886	0.799918	11
7	0.179	0.018	0.161	1317.306222	1316.565040	0.741182	5
8	0.146	0.018	0.128	8427.741041	8426.279792	1.461249	7
9	0.213	0.018	0.195	435271.206036	435270.749052	0.456984	27
10	0.116	0.018	0.098	13383.550396	13383.263840	0.286556	5
min	0.111	0.013	0.092	1252.276319	1250.901896	0.286556	5
avg	0.157	0.017	0.140	106545.9843	106544.1763	1.808045	23
max	0.248	0.019	0.231	435271.206	435270.7491	7.181029	77

Για 1001 μεταβλητές απόφασης, δηλαδή 1000 μεταβλητές x_i και τη μεταβλητή r, φαίνεται ο συγκεκριμένος αλγόριθμος να είναι πολύ πιο γρήγορος, φτάνοντας να είναι ακόμα και 10 φορές πιο γρήγορος. Επίσης, παρατηρείται ότι με την αύξηση της τιμής της αντικειμενικής αυξάνεται και ο αριθμός των απαιτούμενων κόμβων. Η αύξηση της τιμής της αντικειμενικής οφείλεται στην αύξηση των ορίων των περιορισμών, τα οποία παράγονται τυχαία από τη συνάρτηση *insert_data*.

Για N = 5000

Πίνακας 39 **Αποτελέσματα των 10 υπολογιστικών πειραμάτων για 5000 x_i μεταβλητές.**

α/α	Χρόνος (sec)			Τιμές αντικειμενικών			Κόμ-βοι
	CPLEX	Αλγό-ριθμος	Από-κλιση	Γραμμική χαλάρωση	Τελική ακέραια λύση	Απόκλιση	
1	0.292	0.23	0.062	243541.040725	243540.433523	0.607202	15
2	0.23	0.228	0.002	40901.849124	40900.959212	0.889912	7
3	0.254	0.228	0.026	82262.664936	82259.438042	3.226894	19
4	0.234	0.217	0.017	1405.894837	1402.765074	3.129763	5
5	0.347	0.280	0.067	24095.096665	24090.241392	4.855273	13
6	0.235	0.234	0.001	118997.199991	118996.762210	0.437781	11
7	0.242	0.234	0.008	13398.458000	13396.974514	1.483486	7
8	0.255	0.245	0.010	1672.554476	1671.583146	0.971330	5
9	0.327	0.236	0.091	3951.902674	3946.680265	5.222409	11
10	0.284	0.242	0.042	257005.223263	256998.917809	6.305454	75
min	0.23	0.217	0.001	1405.894837	1402.765074	0.437781	5
avg	0.270	0.237	0.033	78723.18847	78720.47552	2.712950	17
max	0.347	0.280	0.091	257005.2233	256998.9178	6.305454	75

Με την αύξηση του αριθμού των μεταβλητών εμφανίζεται αισθητή μείωση της απόκλισης των χρόνων που απαιτεί ο αλγόριθμος και η CPLEX για την επίλυση των προβλημάτων. Παρόλα αυτά ο αλγόριθμος που παρουσιάζεται στη συγκεκριμένη εργασία ξεπερνάει σε ταχύτητα ακόμα τη CPLEX. Τέλος, αξίζει να σημειωθεί ότι και εδώ η αύξηση της βέλτιστης λύσης προκαλεί μια αύξηση και στον αριθμό των απαιτούμενων κόμβων χωρίς να υπάρχει μια απόλυτη σχέση που να συνδέει τις μεταβολές αυτές.

Για N = 10000

Πίνακας 40 **Αποτελέσματα των 10 υπολογιστικών πειραμάτων για 10000 x_i μεταβλητές.**

α/α	Χρόνος (sec)			Τιμές αντικειμενικών			Κόμ-βοι
	CPLEX	Αλγό-ριθμος	Από-κλιση	Γραμμική χαλάρωση	Τελική ακέραια λύση	Απόκλιση	
1	0.452	0.897	-0.445	482948.306782	482946.185490	2.121292	129
2	0.721	0.879	-0.158	37811.714751	37806.592246	5.122505	17
3	0.421	0.884	-0.463	92412.816064	92407.671507	5.144557	11
4	0.501	0.848	-0.347	1278.011928	1277.202095	0.809833	5
5	0.411	0.872	-0.461	113400.581057	113398.281170	2.299887	15
6	0.474	1.135	-0.661	86627.273815	86626.510443	0.763372	9
7	0.421	0.893	-0.472	11167.439770	11162.205961	5.233809	7
8	0.444	0.971	-0.527	386396.555744	386394.374063	2.181681	253
9	0.605	0.878	-0.273	8499.621563	8495.411304	4.210259	13
10	0.473	0.872	-0.399	26031.338319	26023.959215	7.379104	25
min	0.411	0.848	-0.158	1278.011928	1277.202095	0.763372	5
avg	0.492	0.913	-0.4206	124657.366	124653.8393	3.526630	49
max	0.721	1.135	-0.661	482948.3068	482946.1855	7.379104	253

Για 10000 x_i μεταβλητές η CPLEX πλέον ξεπερνάει τον αλγόριθμο, αλλά με διαφορά που οριακά ξεπερνάει το μισό δευτερόλεπτο. Όπως και στα προηγούμενα παραδείγματα έτσι και εδώ η αύξηση της τιμής της αντικειμενικής, των ορίων των περιορισμών δηλαδή, οδηγεί σε αύξηση τον αριθμό των απαιτούμενων κόμβων, χωρίς αυτό να είναι απόλυτο.

Για N = 15000

Πίνακας 41 **Αποτελέσματα των 10 υπολογιστικών πειραμάτων για 15000 x_i μεταβλητές.**

α/α	Χρόνος (sec)			Τιμές αντικειμενικών			Κόμ-βοι
	CPLEX	Αλγό-ριθμος	Από-κλιση	Γραμμική χαλάρωση	Τελική ακέραια λύση	Απόκλιση	
1	0.822	1.985	-1.163	586120.963315	586118.712589	2.250726	155
2	1.234	2.394	-1.160	34169.490655	34162.881408	6.609247	37
3	0.682	1.917	-1.235	101579.351199	101579.256635	0.094564	9
4	0.627	1.877	-1.250	1223.042054	1222.035765	1.006289	5
5	0.666	1.85	-1.184	26009.391041	26004.494882	4.896159	15
6	0.616	2.014	-1.398	629183.311758	629181.939359	1.372399	197
7	0.613	1.933	-1.320	9802.752452	9799.094903	3.657549	7
8	0.57	1.907	-1.337	309200.612369	309200.593965	0.018404	17
9	0.543	1.906	-1.363	80276.182642	80275.275204	0.907438	9
10	0.562	1.939	-1.377	1709.560895	1707.790832	1.770063	5
min	0.543	1.85	-1.160	1223.042054	1222.035765	0.018404	5
avg	0.694	1.972	-1.279	177927.4658	177925.2076	2.258284	46
max	1.234	2.394	-1.398	629183.3118	629181.9394	6.609247	197

Καθώς αυξάνεται ο αριθμός των x_i μεταβλητών αυξάνεται και η απόκλιση μεταξύ των χρόνων των δύο αλγορίθμων επίλυσης, με τη CPLEX να κάνει σχεδόν το μισό χρόνο σε σχέση με το αλγόριθμο της εργασία. Η αύξηση των απαιτούμενων κόμβων φαίνεται να συνδέεται και εδώ με την τιμή της αντικειμενικής.

Για N = 20000

Πίνακας 42 **Αποτελέσματα των 10 υπολογιστικών πειραμάτων για 20000 x_i μεταβλητές.**

α/α	Χρόνος (sec)			Τιμές αντικειμενικών			Κόμ-βοι
	CPLEX	Αλγό-ριθμος	Από-κλιση	Γραμμική χαλάρωση	Τελική ακέραια λύση	Απόκλιση	
1	0.812	4.188	-3.376	704800.556807	704798.690234	1.866573	1267
2	0.922	3.354	-2.432	28980.648868	28978.944639	1.704229	11
3	0.837	3.454	-2.617	110200.1267	110196.3436	3.783143	21
4	0.736	3.322	-2.586	1303.228562	1303.176374	0.052188	3
5	0.746	3.337	-2.591	170448.366657	170446.154571	2.212086	19
6	0.976	3.373	-2.397	3881.046646	3877.456752	3.589894	9
7	0.716	3.358	-2.642	24663.934615	24661.936833	1.997782	7
8	0.77	3.342	-2.572	1372.314921	1370.373439	1.941482	5
9	0.925	3.985	-3.060	10159.695132	10157.161263	2.533869	7
10	0.795	3.348	-2.553	63301.931721	63299.899179	2.032542	9
min	0.716	3.322	-2.397	1303.228562	1303.176374	0.052188	3
avg	0.824	3.506	-2.683	111911.1851	111909.0137	2.171379	136
max	0.976	4.188	-3.376	704800.5568	704798.6902	3.783143	1267

Για 20001 μεταβλητές, 20000 x_i και την r, η απόκλιση στους χρόνους επίλυσης αυξάνεται κατά πολύ σε σχέση με τα προηγούμενα παραδείγματα και σχεδόν τετραπλασιάζεται. Ο αριθμός των κόμβων επίλυσης και στα συγκεκριμένα προβλήματα φαίνεται να επηρεάζεται από τα όρια των περιορισμών, που μεταφράζονται ως βέλτιστη τιμή στα παραπάνω παραδείγματα.

Συνοπτικά αποτελέσματα

Πίνακας 43 **Συνοπτικά αποτελέσματα των 50 υπολογιστικών πειραμάτων που επιλύθηκαν στα πλαίσια της εργασίας.**

Αριθμός μεταβλητών N			1000	5000	10000	15000	20000
Χρόνος (sec)	CPLEX	min	0.111	0.23	0.411	0.543	0.716
		avg	0.157	0.270	0.492	0.694	0.824
		max	0.248	0.347	0.721	1.234	0.976
	Αλγό-ριθμος	min	0.013	0.217	0.848	1.85	3.322
		avg	0.017	0.237	0.913	1.972	3.506
		max	0.019	0.280	1.135	2.394	4.188
	από-κλιση	min	0.092	0.001	-0.158	-1.160	-2.397
		avg	0.140	0.033	-0.421	-1.279	-2.683
		max	0.231	0.091	-0.661	-1.398	-3.376
Απόκλιση τιμών αντικειμενικών		min	0.286556	0.437781	0.763372	0.018404	0.052188
		avg	1.808045	2.712950	3.526630	2.258284	2.171379
		max	7.181029	6.305454	7.379104	6.609247	3.783143
Κόμβοι		min	5	5	5	5	3
		avg	23	17	49	46	136
		max	77	75	253	197	1267

Από τα συνοπτικά αποτελέσματα του παραπάνω πίνακα φαίνεται ότι μέχρι και τις 5000 μεταβλητές ο συγκεκριμένος αλγόριθμος είναι γρηγορότερος της CPLEX, ενώ στις 20000 μεταβλητές η μέση απόκλιση ξεπέρασε τα 2 δευτερόλεπτα υπέρ της CPLEX. Όσον αφορά την απόκλιση των αντικειμενικών τιμών μεταξύ της γραμμικής χαλάρωσης και της ακέραιας

λύσης φαίνεται να μην επηρεάζεται από τον αριθμό των μεταβλητών και είναι συνεχώς πολύ μικρή σε όλες περιπτώσεις, ειδικά αν ληφθεί υπόψη ότι οι τιμές των αντικειμενικών μπορεί να φτάσουν και μερικές εκατοντάδες χιλιάδες.

7. Συμπεράσματα – Μελλοντική έρευνα

Στην εργασία αυτή μελετήθηκε μία περίπτωση ακέραιου προβλήματος σακιδίου με περιορισμούς πολλαπλών επιλογών, με εφαρμογή σε μία εφοδιαστική αλυσίδα. Αναλύθηκε ο βέλτιστος τρόπος εφαρμογής τιμών εκποίησης σε μία προκαθορισμένη χρονική περίοδο και με συγκεκριμένο διαθέσιμο απόθεμα, με σκοπό τη μεγιστοποίηση των εσόδων. Για την επίλυση αναπτύχθηκε ένας αλγόριθμος branch and bound ο οποίος χρησιμοποιώντας τις ιδιότητες του προβλήματος και τη σχέση μεταξύ γειτονικών κόμβων έκανε εφικτή την γρήγορη επίλυση των γραμμικών χαλαρώσεων των κόμβων του δέντρου. Στη συνέχεια αναπτύχθηκε και εκτελέστηκε κώδικας σε γλώσσα C και εκπονήθηκαν μικρά και διαχειρίσιμα πειράματα ώστε να εξασφαλιστεί η ορθότητά του. Έπειτα ο κώδικας δοκιμάστηκε με διαφορετικούς αριθμούς μεταβλητών ώστε να συγκριθεί η αποτελεσματικότητά του με αυτή του λογισμικού βελτιστοποίησης CPLEX μέσω του χρόνου που χρειάζονταν για να βρουν την ακέραια λύση. Από τη σύγκριση αυτή έγινε ξεκάθαρο ότι για μικρό αριθμό μεταβλητών ο αλγόριθμος ήταν κατά πολύ γρηγορότερος της CPLEX, για μεταβλητές όμως της τάξης των 10000 και άνω οι διαφορές ήταν αισθητές ανάμεσα στις δύο μεθόδους με τον αλγόριθμο να κάνει πολλές φορές πάνω από το διπλάσιο χρόνο της CPLEX.

Τέλος, λίγο πριν την ολοκλήρωση της εργασίας θα προταθούν κάποια θέματα τα οποία θα μπορούσαν να αποτελέσουν έμπνευση για περαιτέρω έρευνα πάνω στο θέμα.

Στα πλαίσια της διπλωματικής αυτής ο αλγόριθμος και η αποτελεσματικότητά του δοκιμάστηκαν μέσα από μια σειρά πειραμάτων που έφτασαν μέχρι και τις 20000 μεταβλητές και παρότι εξήχθησαν σημαντικά αποτελέσματα για αυτόν, μια πιο κατάλληλη δοκιμασία για την απόδοσή του θα ήταν η σύγκριση με άλλους εξειδικευμένους αλγόριθμους. Ακόμη, ο αλγόριθμος αυτός θα μπορούσε να τροποποιηθεί ώστε να έχει εφαρμογή σε μεγαλύτερο πεδίο προβλημάτων, ακόμα και αυτών με επιπρόσθετους περιορισμούς. Για να γίνει όμως αυτό θα πρέπει να μελετηθούν παραπάνω οι θεωρητικές ιδιότητες πάνω στις οποίες βασίστηκε ο αλγόριθμος.

Βιβλιογραφία

Aggarwal, V., Deo, N., & Sarkar, D. (1992). The knapsack problem with disjoint multiple-choice constraints. *Naval Research Logistics*(39), pp. 213-227.

Armstrong, R., Kung, D., Sinha, P., & Zoltners, A. (1983). A computational study of a multiple-choice knapsack algorithm. *ACM Transactions on Mathematical Software*(9), pp. 184-198.

Dyer, M., Kayal, N., & Walker, J. (1984). A branch and bound algorithm for solving the multiple-choice knapsack problem. *Journal of Computational and Applied Mathematics,*(11), 231-249.

Dyer, M., Riha, W., & Walker, J. (1995). A hybrid dynamic programming/branch and bound algorithm for the multiplechoice knapsack problem. *Journal of Computational and Applied Mathematics*(58), pp. 43-54.

Glover, F., & Klingman, D., (1979). An O(n log n) algorithm for LP knapsacks with GUB constraints. Mathematical Programming (17), pp. 345–361.

Kozanidis, G., & Melachrinoudis, E. (2004). A branch & bound algorithm for the 0-1 mixed integer knapsack problem with linear multiple choice constraints. *Computers and Operations Research*(31), pp. 695-711.

Lin, E. Y.-H. (1998a). Multiple Choice Knapsack Problems and its Extensions on Capital Investment. In D.-Z. Du, X.-S. Zhang, & K. Cheng, *Operations Research and its Applications* (pp. 406 - 417). Πεκίνο.

Lin, E.Y.-H. (1998b) A bibliographical survey on some well known non-standard knapsack problems. INFOR (36), pp. 274–317.

Nauss, R. M. (1978). The 0-1 knapsack problem with multiple choice constraints. *European Journal of Operational*(2), pp. 125-131.

Pisinger, D. (1995). A minimal algorithm for the multiple choice knapsack problem. *European Journal of Operations*(83), pp. 394-410.

Rouf, M. (2005). *Incremental Convex Hull Approach Applied to Optimal Admission Control & QoS Adaptation in Multimedia Systems.* Ντάκα.

Sarin, S., Karwan, M.H., (1989). The linear multiple choice knapsack problem. Operations Research Letters (8), pp. 95–100.

Sinha, P., & Zoltners, A. (1979). A minimal algorithm for the multiple choice knapsack problem. *European Journal of Operational*(27), pp. 503-515.

Zaarour, N. Z. (2011). *Phase-out and disposal issues of obsolete inventory items in retail stores.* Βοστώνη.

Κοζανίδης, Γ. (2008, Σεπτέμβριος). Ακέραιος Προγραμματισμός & Συνδυαστική Βελτιστοποίηση. Βόλος: Εκδόσεις Πανεπιστημίου Θεσσαλίας.

Παπαγεωργίου, Βερίλλης, Κοζανίδης (2015) «Ένας αλγόριθμος branch and bound για μια ειδική περίπτωση του ακέραιου προβλήματος σακιδίου με περιορισμούς πολλαπλών επιλογών». 4° φοιτητικό συνέδριο της Ελληνικής Εταιρείας Επιχειρησιακής Έρευνας, Αθήνα, 17-18 Δεκεμβρίου

ΠΑΡΑΡΤΗΜΑ

ΚΩΔΙΚΑΣ

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <math.h>
#include <ilcplex/ilocplex.h>


//-----insert R[N+1], V[N+1]-----
#define N 1000
double I = 33.5;
double T = 10;
double R[N+1] = {2, 4, 7, 9, 10};
double V[N+1] = {1, 1, 2, 4, 6};
double OV = 0;

//#define user_input

// Max 4X1 + 7X2 + 9X3+ 10X4 + 2R
// s.t. X1 + 2X2 + 4X3 + 6X4 + R <= 33.5
//     X1 +  X2 +  X3 + 6 X4 <= 10


CPXENVptr   Env = NULL;  // CPLEX environment pointers
CPXLPptr    Lp = NULL;   // CPLEX problem pointers
double     *rmatval;
int        treeindex, *rmatind, rmatbeg = 0;
```

```
FILE *myfile;

typedef struct VarNode{
        int    index;
        struct VarNode* right;
        struct VarNode* left;
} VNode;

typedef struct TreeNode{
        int    index;
        double  LpObj, Ires, Tres, FracVal;
        double  UBound[N+1], LBound[N+1], Value[N+1];
        VNode   *Current;
        struct  TreeNode* right;
        struct  TreeNode* left;
} TNode;

VNode *VarList = NULL;
TNode *TreeList = NULL;

double min(double a, double b)
{
        if (a < b) return a;
        else return b;
}
```

```
void insert_node_to_tail(int in) //adds node at the end of the list (used for x-variables)
{
	VNode *Current, *Temp;

	Temp = (VNode*)malloc(sizeof(VNode));
	Temp->index = in;
	Temp->right = NULL;
	Temp->left = NULL;

	if (VarList == NULL)
		VarList = Temp;
	else{
		Current = VarList;
		while (Current->right != NULL)
			Current = Current->right;
		Current->right = Temp;
		Temp->left = Current;
	}
}


void insert_node_in_place() //adds node in correct position of the list (used for R-var)
{
	VNode  *Temp, *Prev, *Next;
	double Rratio;
```

```
Rratio = R[0]/V[0];


Temp = (VNode*)malloc(sizeof(VNode));
Temp->index = 0;
Temp->right = NULL;
Temp->left = NULL;


if (VarList == NULL)
        VarList = Temp;
else{
        if (R[VarList->index]/V[VarList->index] <= Rratio){ // in this case, item will be
inserted in the first position of the list
                Temp->right = VarList;
                VarList->left = Temp;
                VarList = Temp;
        }
        else{  // in this case, item will be inserted after first position
                Prev = VarList;
                Next = Prev->right;
                while (Next != NULL && (R[Next->index]-R[Prev->index])/(V[Next-
>index]-V[Prev->index]) > Rratio){
                        Prev = Next;
                        Next = Next->right;
                }
                if (Next == NULL){  // in this case, Temp is inserted at the end of the
list (there is no node after Prev)
                        Prev->right = Temp;
                        Temp->left = Prev;
                }
```

```
                else{  // in this case, Temp is inserted between Prev and Next
                        Prev->right = Temp;
                        Temp->right = Next;
                        Next->left = Temp;
                        Temp->left = Prev;
                }
        }
    }
}


void InsertTreeNode(int index, double lpobj, double ires, double tres, double FracValue,
double UBound[], double LBound[],
                double Value[], VNode  *Current) //adds node in correct position of the
tree-list
{
    int  ii;
    TNode  *Temp, *Prev, *Next;

    Temp = (TNode*)malloc(sizeof(TNode));
    Temp->index = index;
    Temp->LpObj = lpobj;
    Temp->Ires = ires;
    Temp->Tres = tres;
    Temp->FracVal = FracValue;
    for (ii = 0; ii <= N; ii++){
        Temp->UBound[ii] = UBound[ii];
        Temp->LBound[ii] = LBound[ii];
```

```
            Temp->Value[ii] = Value[ii];
        }
        Temp->Current = Current;
        Temp->right = NULL;
        Temp->left = NULL;

        if (TreeList == NULL)
            TreeList = Temp;
        else{
            if (TreeList->LpObj <= Temp->LpObj){ // in this case, item will be inserted in
the first position of the list
                Temp->right = TreeList;
                TreeList->left = Temp;
                TreeList = Temp;
            }

            else{  // in this case, item will be inserted after first position
                Prev = TreeList;
                Next = Prev->right;
                while (Next != NULL && (Next->LpObj > Temp->LpObj)){
                    Prev = Next;
                    Next = Next->right;
                }
                if (Next == NULL){  // in this case, Temp is inserted at the end of the
list (there is no node after Prev)
                    Prev->right = Temp;
                    Temp->left = Prev;
                }
```

```
                else{ // in this case, Temp is inserted between Prev and Next
                        Prev->right = Temp;
                        Temp->right = Next;
                        Next->left = Temp;
                        Temp->left = Prev;
                }
        }
    }
}


void DeleteTreeNode(TNode **Leaving) //deletes node from tree-list
{
    TNode  *Temp;
    if (*Leaving == TreeList){
        Temp = TreeList;
        TreeList = TreeList->right;
    }
    else{
        Temp = *Leaving;
        if (Temp->left != NULL)
            Temp->left->right = (*Leaving)->right;
        if  (Temp->right != NULL)
            Temp->right->left = (*Leaving)->left;
    }
    free(Temp);
}
```

```
void make_list() //makes list L based on each var's ratio
{
        int i, index;
        double ratio, largest;
        double lastR, lastV;

        index = -2;
        largest = 0;
        lastR = 0;
        lastV = 0;

        while (index != -1) {
                index = -1;
                for (i = 1; i <= N; i ++){
                        if (R[i] <= lastR) continue;
                        ratio = (R[i] - lastR) / (V[i] - lastV);
                        if (ratio > largest) {
                                largest = ratio;
                                index = i;
                        }
                }

                if (index != -1){
                        insert_node_to_tail(index);
                        lastR = R[index];
```

```
                    lastV = V[index];
                }

                largest = 0;
            }

            insert_node_in_place();
}


void insert_data() //makes R[N+1] and V[N+1] using random numbers
{

#ifndef user_input
            int i;
            double random,alpha, beta, C, max = 0;
            double P[N];

            srand(time(NULL));

            random = (double)(rand())/(double)RAND_MAX;

            alpha = (random * 4000) + 1000;
            beta = (random * 3) - 4;
            C = (random * 9) + 1;
            T = ceil((random * 24) + 12);
```

```
        for (i = 0; i <= N-1; i ++){
                P[i] = C + ((random * 4) + 1) * (N - i);
        }

        for (i = 1; i <= N; i ++) {
                R[i] = alpha*pow(P[i-1],beta + 1);
                V[i] = alpha*pow(P[i-1],beta);
        }

        R[0] = C;
        V[0] = 1;

        I = 0;
        for (i = 0; i <=N; i ++){
                I += V[i];
        }
        I *= 15;

#endif
}
```

```
// finds the first var x to the left of Temp that can be decreased (is greater than its lower bound)
VarNode *GetPrevDecrX(VarNode *Temp, double LBound[], double Value[]){
        if (Temp != NULL) Temp = Temp->left;
        while (Temp != NULL && (Temp->index == 0 || Value[Temp->index] <= LBound[Temp->index]))
```

```
                Temp = Temp->left;
        return Temp;
}



// finds the first var x to the left of Temp that can be increased (is smaller than its upper
bound)
VarNode *GetPrevIncrX(VarNode *Temp, double UBound[], double Value[]){
        if (Temp != NULL) Temp = Temp->left;
        while (Temp != NULL && (Temp->index == 0 || Value[Temp->index] >=
UBound[Temp->index]))
                Temp = Temp->left;
        return Temp;
}



// finds the first var x to the right of Temp that can be increased (is smaller than its upper
bound)
VarNode *GetNextIncrX(VarNode *Temp, double UBound[], double Value[]){
        if (Temp != NULL) Temp = Temp->right;
        while (Temp != NULL && (Temp->index == 0 || Value[Temp->index] >=
UBound[Temp->index]))
                Temp = Temp->right;
        return Temp;
}



void objective_value(TreeNode *BestNode) //used to find obj value in kozalg
{
```

```
        int i;


        for (i = 0; i <= N; i ++){
                OV += R[i] * BestNode->Value[i];
        }
}


// allocates Ires optimally by increasing variables in the list starting from Current and going to the right
void GoRight(int t, double *LpObj, double *Ires, double *Tres, double *FracValue, double UBound[], double LBound[], double Value[], VarNode **Current)
{
        double ires, tres, fracvalue, lpobj, temp, a;
        VNode *Prev, *Next;


        ires = *Ires;
        tres = *Tres;
        lpobj = *LpObj;
        fracvalue = *FracValue;


        Next = *Current;


        while (Next != NULL && ires > 0.00000001) {
                if (Value[Next->index] >= UBound[Next->index]){  // cannot increase vars at their upper bound
                        Next = Next->right;
                        continue;
                }
```

```
if (Next->index == 0){  // if next var is R
        fracvalue = min(ires/V[0], UBound[0] - Value[0]);
        ires = ires - fracvalue*V[0];
        lpobj = lpobj + fracvalue*R[0];
        Value[0] = Value[0] + fracvalue;
        if (ires > 0.00000001 && Value[0] >= UBound[0]){
            Next = Next->right;
        }
    }
    else if (tres > 0.00000001){
        fracvalue =min(ires/V[Next->index], min(UBound[Next->index]-
Value[Next->index], tres));
        ires = ires - fracvalue*V[Next->index];
        tres = tres - fracvalue;
        lpobj = lpobj + fracvalue*R[Next->index];
        Value[Next->index] = Value[Next->index] + fracvalue;

        if (ires > 0.00000001 && Value[Next->index] >= UBound[Next-
>index]){
            Next = Next->right;
        }
    }
    else{  // in this case, ires > 0, but tres = 0
        Prev = GetPrevDecrX(Next, LBound, Value);
        if (Prev == NULL) {
            break;
        }
        if (ires/(V[Next->index]-V[Prev->index]) < min(UBound[Next->index]
- Value[Next->index], Value[Prev->index] - LBound[Prev->index])){
```

```
                        fracvalue = ires/(V[Next->index]-V[Prev->index]);
                        //ires = 0;
                        ires = ires - fracvalue*(V[Next->index]-V[Prev->index]);
                        lpobj = lpobj + fracvalue*(R[Next->index]-R[Prev->index]);
                        Value[Next->index] = Value[Next->index] + fracvalue;
                        Value[Prev->index] = Value[Prev->index] - fracvalue;
                    }
                    else if (UBound[Next->index] - Value[Next->index] <= Value[Prev-
>index] - LBound[Prev->index]){
                        fracvalue = UBound[Next->index] - Value[Next->index] ;
                        ires = ires - fracvalue*(V[Next->index]-V[Prev->index]);
                        lpobj = lpobj + fracvalue*(R[Next->index]-R[Prev->index]);
                        Value[Next->index] = UBound[Next->index];
                        Value[Prev->index] = Value[Prev->index] - fracvalue;
                        Next = Next->right;
                    }
                    else{
                        fracvalue = Value[Prev->index] - LBound[Prev->index];
                        ires = ires - fracvalue*(V[Next->index]-V[Prev->index]);
                        lpobj = lpobj + fracvalue*(R[Next->index]-R[Prev->index]);
                        Value[Next->index] = Value[Next->index] + fracvalue;
                        Value[Prev->index] = LBound[Prev->index];
                    }
                }
            }

            if (Next != NULL) fracvalue = Value[Next->index];
            else fracvalue = 1;
```

```
        *LpObj = lpobj;
        *Ires = ires;
        *Tres = tres;
        *FracValue = fracvalue;
        *Current = Next;
}

// de-allocates Ires optimally by decreasing variables in the list starting from Current and going to the left
void GoLeft(int t,double *LpObj, double *Ires, double *Tres, double *FracValue, double UBound[], double LBound[], double Value[], VarNode **Current)
{
        double ires, tres, fracvalue, lpobj, temp;
        int i;
        VNode *Prev, *Next, *Prev1;

        ires = *Ires;
        tres = *Tres;
        lpobj = *LpObj;
        fracvalue = *FracValue;
        Prev1 = NULL;

        Next = *Current;

        while (Next != NULL && ires < -0.00000001) {
                if (Value[Next->index] <= LBound[Next->index]){
```

```
                Next = Next->left;
                continue;
        }
        if (fracvalue == -1) break;
        if (Next->index == 0){
                if (tres < 0){
                        fracvalue = -1;
                        break;
                }
                fracvalue = min(-ires/V[0], Value[Next->index] - LBound[Next-
>index]);
                ires = ires + (fracvalue*V[0]);
                lpobj = lpobj - (fracvalue*R[0]);
                Value[0] = Value[0] - fracvalue;
                if (ires < -0.00000001 && Value[0] <= LBound[0])
                        Next = Next->left;
        }
        else if (tres > 0.00000001){
                fracvalue = min(-ires/V[Next->index], Value[Next->index] -
LBound[Next->index]);
                ires = ires + (fracvalue*V[Next->index]);
                tres = tres + fracvalue;
                lpobj = lpobj - (fracvalue*R[Next->index]);
                Value[Next->index] = Value[Next->index] - fracvalue;
                if (ires < -0.00000001 && Value[Next->index] <= LBound[Next-
>index])
                        Next = Next->left;
        }
        else{  // in this case, ires < 0, but tres = 0
```

```
                Prev = GetPrevIncrX(Next, UBound, Value);
                if (Prev == NULL){
                        Next = Next->right;
                        Prev1 = GetPrevDecrX(Next, LBound, Value); //finds the var
that can be decreased
                        Next = Next->left;
                        if (Prev1 == NULL){ // problem is infeasible in this case, no
var available for decrease
                                fracvalue = -1;
                                break;}
                        else {
                                fracvalue = min(-ires/V[Prev1->index], Value[Prev1-
>index] - LBound[Prev1->index]);
                                ires = ires + (fracvalue*V[Prev1->index]);
                                Value[Prev1->index] = Value[Prev1->index] -
fracvalue;
                                lpobj = lpobj - (fracvalue*R[Prev1->index]);
                                if (ires < -0.00000001 && Value[Prev1->index] <=
LBound[Prev1->index]) Next = Next->left;
                        }
                }
                else if (-ires/(V[Next->index] - V[Prev->index]) < min(Value[Next-
>index] - LBound[Next->index], UBound[Prev->index] - Value[Prev->index])){
                        fracvalue = -ires/(V[Next->index]-V[Prev->index]);
                        ires = 0;
                        lpobj = lpobj - (fracvalue*(R[Next->index]-R[Prev->index]));
                        Value[Next->index] = Value[Next->index] - fracvalue;
                        Value[Prev->index] = Value[Prev->index] + fracvalue;
                }
```

```
                    else if (Value[Next->index] - LBound[Next->index] <= UBound[Prev-
>index] - Value[Prev->index]){
                        fracvalue = Value[Next->index] - LBound[Next->index];
                        ires = ires + (fracvalue*(V[Next->index]-V[Prev->index]));
                        lpobj = lpobj - (fracvalue*(R[Next->index]-R[Prev->index]));
                        Value[Next->index] = LBound[Next->index];
                        Value[Prev->index] = Value[Prev->index] + fracvalue;
                        Next = Next->left;
                    }
                    else{
                        fracvalue = UBound[Prev->index] - Value[Prev->index];
                        ires = ires + fracvalue*(V[Next->index]-V[Prev->index]);
                        lpobj = lpobj - fracvalue*(R[Next->index]-R[Prev->index]);
                        Value[Next->index] = Value[Next->index] - fracvalue;
                        Value[Prev->index] = UBound[Prev->index];

                    }
                }
            }
            if (Next != NULL && Prev1 != NULL) fracvalue = Value[Prev1->index];
            else if (Next != NULL && fracvalue != -1) fracvalue = Value[Next->index];
            else fracvalue = -1;  // signifies that problem is infeasible
            *LpObj = lpobj;
            *Ires = ires;
            *Tres = tres;
            *FracValue = fracvalue;
            *Current = Next;
}
```

```
void dsp_cpx_error(char *name)
{
  printf("%s failed, exiting...\n", name);
  printf("Press return to continue...\n");
  getchar();
}


bool SetIntParam(CPXENVptr env, int whichparam, int value)
{
   int status;
   status = CPXsetintparam(env, whichparam, value);
   if (status != 0){
    dsp_cpx_error("CPXsetintparam");
    return false;
   }
   return true;
}


bool init_env(CPXENVptr *env)
{
  int   status;
  *env = CPXopenCPLEX(&status);
  if (status != 0){
```

```
    dsp_cpx_error("CPXopenCPLEX");
    return false;
  }
  if (SetIntParam(*env, CPX_PARAM_DATACHECK, CPX_ON) == false) return false;  // when
on, this setting performs various data checks during CPLEX function execution
  return true;
}



bool init_lp(CPXENVptr env, CPXLPptr *lp, char *name)
{
  int   status;

  *lp = CPXcreateprob(env, &status, name);
  if (status != 0){
    dsp_cpx_error("CPXcreateprob");
    return false;
  }
  return true;
}



bool NewCols (CPXCENVptr env, CPXLPptr lp, int ccnt, double const * obj, double const * lb,
double const * ub, char const * xctype, char ** colname)
{
  int status;
  status = CPXnewcols (env, lp, 1, obj, lb, ub, xctype, NULL);
  if (status != 0){
```

```
    dsp_cpx_error("CPXnewcols");
    return false;
  }
  return true;
}


bool FreeProb (CPXENVptr env, CPXLPptr *lp)
{
  int status;
  status = CPXfreeprob (env, lp);
  if (status != 0){
    dsp_cpx_error("CPXfreeprob");
    return false;
  }
  return true;
}


bool FreeEnv (CPXENVptr *env)
{
  int status;
  status = CPXcloseCPLEX (env);
  if (status != 0){
    dsp_cpx_error("CPXcloseCPLEX");
    return false;
  }
```

```
  return true;
}


bool GetObjVal(CPXENVptr env, CPXLPptr lp, double *objval)
{
  int status;
  status = CPXgetobjval (env, lp, objval);
  if (status != 0){
    dsp_cpx_error("CPXgetobjval");
    return false;
  }
  return true;
}


bool LPOpt (CPXENVptr env, CPXLPptr lp)
{
  int status;
  status = CPXlpopt (env, lp);
  if (status != 0){
    dsp_cpx_error("CPXlpopt");
    return false;
  }
  return true;
}
```

```
bool MipOpt(CPXENVptr env, CPXLPptr lp)
{
        int status;
        status = CPXmipopt (env, lp);
        if (status != 0){
                dsp_cpx_error("CPXmipopt");
                return false;
        }
        return true;
}


bool Getx(CPXCENVptr env, CPXCLPptr lp, double *sol)
{
  int status, numcols;
  numcols = CPXgetnumcols(env,lp);
  status = CPXgetx (env,lp,sol,0,numcols-1);
  if (status != 0){
    dsp_cpx_error("CPXgetx");
    return false;
  }
  return true;
}


bool PrintProblem (CPXENVptr env, CPXLPptr lp, char const * filename)
```

```
{
  int status;

  status = CPXwriteprob (env, lp, filename, NULL);
  if (status != 0){
    dsp_cpx_error("CPXwriteprob");
    return false;
  }
  return true;
}

bool Addrows(CPXCENVptr env, CPXLPptr lp, int nzcnt, double const * rhs, char const *
sense,
        int const * rmatind, double const * rmatval, char ** rowname)
{
  int  status;
  status = CPXaddrows (env, lp, 0, 1, nzcnt, rhs, sense, &rmatbeg, rmatind, rmatval, NULL,
NULL);
  if (status != 0){
    dsp_cpx_error("CPXaddrows");
    return false;
  }

  return true;
}
```

```
bool  ChangeObjSense(CPXENVptr Env, CPXLPptr Lp, int maxormin)
{
        int  status;
        status = CPXchgobjsen (Env, Lp, maxormin);
        if (status != 0){
    dsp_cpx_error("CPXchgobjsen");
    return false;
  }
  return true;
}


bool  init_problem()
{
  int status;
  Env = CPXopenCPLEX (&status);
  if (status != 0){
    dsp_cpx_error("CPXopenCPLEX");
    return false;
  }

  Lp = CPXcreateprob (Env, &status, "anna");
  if (status != 0){
    dsp_cpx_error("CPXcreateprob");
    return false;
  }
```

```
  status = CPXsetintparam(Env, CPX_PARAM_DATACHECK, CPX_ON);
  if (status != 0){
          dsp_cpx_error("CPXsetintparam");
          return false;
  }

  if (ChangeObjSense(Env, Lp, -1) == false) return false;

  return true;
}

bool free_problem ()
{
   if (FreeProb (Env, &Lp) == false) return false;
   if (FreeEnv(&Env) == false) return false;
   return true;
}

bool ResourceConstraint()
{
  double rhs, *rmatval;
  char *rowname, sense;
  int i, *rmatind;

  rowname = (char*) malloc(5*sizeof(char));
```

```
  rmatval = (double*)malloc((N+1)*sizeof(double));
  rmatind = (int*)malloc((N+1)*sizeof(int));

  rhs = I;
  sense = 'L';
  sprintf(rowname, "RC");

  for (i=0; i<=N; i++){
    rmatind[i] = i;
    rmatval[i] = V[i];
  }
  if (Addrows(Env, Lp, N+1, &rhs, &sense, rmatind, rmatval, &rowname) == false) return
false;

  free(rowname);
  free(rmatval);
  free(rmatind);

  return true;
}

bool MultipleChoiceConstraint()
{
  double rhs, *rmatval;
  char *rowname, sense;
  int i, *rmatind;
```

```
  rowname = (char*) malloc(5*sizeof(char));
  rmatval = (double*)malloc((N)*sizeof(double));
  rmatind = (int*)malloc((N)*sizeof(int));

  rhs = T;
  sense = 'L';
  sprintf(rowname, "MC");
  for (i=0; i<=N-1; i++){
    rmatind[i] = i+1;
        rmatval[i] = 1;
  }
  if (Addrows(Env, Lp, N, &rhs, &sense, rmatind, rmatval, &rowname) == false) return false;

  free(rowname);
  free(rmatval);
  free(rmatind);

  return true;
}

bool AddDecVars()
{
  double obj, lb, ub;
  char xctype;
  int i;
```

```
  lb = 0;
  ub = CPX_INFBOUND;
  xctype = 'I'; //integer

  for (i=0; i<=N; i++){
    obj = R[i];
    if (NewCols(Env, Lp, 1, &obj, &lb, &ub, &xctype, NULL) == false) return false;
  }
  return true;
}


void print_results(TreeNode *BestNode)
{
        int i,ii,numcols;
        double objval, *sol;

        myfile = fopen("results.txt","w+");

        //----------file printing----------

        fprintf(myfile,"\n");
        fprintf(myfile, "   Profit          Cost \n");
        for (i = 0; i <= N; i ++) {
                fprintf(myfile, "%7f%20f\n", R[i], V[i]);
        }
        fprintf(myfile, "\nI = %f \n", I);
```

```
        fprintf(myfile, "T = %f \n \n", T);


        fprintf(myfile, "\n----------Koza-way results---------- \n");
        fprintf(myfile, "\nOptimal obj = %f \n", OV);
        for (i = 0; i <=N; i ++){
                if (BestNode->Value[i] > 0){
                        if (i == 0) fprintf(myfile,"Optimal value of variable   R = %f
\n",BestNode->Value[i]);
                        else fprintf (myfile,"Optimal value of variable X[%d]  = %f \n", i,
BestNode->Value[i]);
                }
        }
        fprintf(myfile, "Tree node index = %d \n", BestNode->index);
        fprintf(myfile, "Total number of nodes = %d \n", treeindex);


        fprintf(myfile, "\n----------CPLEX-way results---------- \n");
        numcols = CPXgetnumcols(Env,Lp);
        sol = (double*) malloc(numcols*sizeof(double));


        if (GetObjVal(Env, Lp, &objval) == false) exit;
        if (Getx(Env, Lp, sol) == false) exit;


        fprintf(myfile, "\nOptimal obj = %Lf \n", objval);
        if (sol[0] > 0)
                fprintf(myfile,"Optimal value of variable   R = %Lf \n", sol[0]);
        for (ii = 1; ii <= numcols-1; ii++){
                if (sol[ii] <= 0) continue;
          fprintf(myfile,"Optimal value of variable X[%d]  = %Lf \n", ii, sol[ii]);
```

```
	}

	fprintf(myfile, "\n");
	fprintf(myfile, "R[N+1]=");
	for (i = 0; i <= N ; i ++){
		fprintf(myfile,"%f,", R[i]);
	}
	fprintf(myfile, "\n");
	fprintf(myfile, "V[N+1]=");
	for (i = 0; i <= N ; i ++){
		fprintf(myfile,"%f,", V[i]);
	}
	fprintf(myfile, "\n");
	fprintf(myfile, "I=%f", I);
	fprintf(myfile, "\n");
	fprintf(myfile, "T=%f", T);

	fclose(myfile);


	//----------screen printing----------
	printf( "\n----------Koza-way results---------- \n \n");
	printf( "\nOptimal obj = %Lf \n", OV);
	for (i = 0; i <= N; i ++){
		if (BestNode->Value[i] > 0){
			if (i == 0) printf("Optimal value of variable   R = %f \n",BestNode-
>Value[i]);
```

```
                    else printf("Optimal value of variable X[%d]  = %f \n", i, BestNode-
>Value[i]);
                }
        }
        printf("Tree node index = %d \n", BestNode->index);
        printf("Total number of nodes = %d \n", treeindex);

        printf( "\n----------CPLEX-way results---------- \n");
        numcols = CPXgetnumcols(Env,Lp);
        sol = (double*) malloc(numcols*sizeof(double));

        if (GetObjVal(Env, Lp, &objval) == false) exit;

        if (Getx(Env, Lp, sol) == false) exit;

        printf( "\nOptimal obj = %Lf \n", objval);
        if (sol[0] > 0)
                printf("Optimal value of variable   R = %Lf \n", sol[0]);
        for (ii = 1; ii <= numcols-1; ii++){
                if (sol[ii] <= 0) continue;
          printf("Optimal value of variable X[%d]  = %Lf \n", ii, sol[ii]);
        }

        free(sol);

}
```

```
void free_list()
{
        VNode *Prev, *Next;

        Prev = VarList;
        while (Prev != NULL){
                Next = Prev->right;
                free(Prev);
                Prev = Next;
        }
        VarList = NULL;
}


void UndoLeft (double *lpobj, double *ires, double *tres, double fracvalue, double
ubound[], double lbound[], double value[], VarNode **Current)
{
        double  LpObj, Ires, Tres, downinteger;
        VarNode *Next, *Prev;

        LpObj = *lpobj;
        Ires = *ires;
        Tres = *tres;
        Next = *Current;

        downinteger = floor(fracvalue);
        ubound[Next->index] = downinteger;
        if (Next->index == 0){
```

```
                Ires = Ires + (fracvalue-downinteger)*V[0];
                LpObj = LpObj - (fracvalue-downinteger)*R[0];
        }
        else if (Tres > 0.00000001){
                Ires = Ires + (fracvalue-downinteger)*V[Next->index];
                Tres = Tres + (fracvalue-downinteger);
                LpObj = LpObj - (fracvalue-downinteger)*R[Next->index];
        }
        else{  // in this case, ires > 0, but tres = 0
                Prev = GetPrevIncrX(Next, ubound, value);
                if (Prev != NULL){
                        Ires = Ires + (fracvalue-downinteger)*(V[Next->index]-V[Prev-
>index]);
                        LpObj = LpObj - (fracvalue-downinteger)*(R[Next->index]-R[Prev-
>index]);
                        value[Prev->index] = value[Prev->index] + fracvalue - downinteger;
                }
                else{
                        Ires = Ires + (fracvalue - downinteger)*V[Next->index];
                        Tres = Tres + (fracvalue - downinteger);
                        LpObj = LpObj - (fracvalue-downinteger)*R[Next->index];
                }
        }
        value[Next->index] = downinteger;
        Next = Next->right;

        *lpobj = LpObj;
        *ires = Ires;
```

```
        *tres = Tres;
        *Current = Next;
}


void FinishRight (double *lpobj, double *ires, double *tres, double *fracvalue, double
ubound[], double lbound[], double value[], VarNode **Current)
{
        double  LpObj, Ires, Tres, upinteger, FracValue;
        VarNode *Next, *Prev;

        LpObj = *lpobj;
        Ires = *ires;
        Tres = *tres;
        Next = *Current;
        FracValue = *fracvalue;

        upinteger = ceil(FracValue);
        lbound[Next->index] = upinteger;
        if (Next->index == 0){
                Ires = Ires - (upinteger-FracValue)*V[0];
                LpObj = LpObj + (upinteger-FracValue)*R[0];
        }
        else if (Tres > 0.00000001){
                Ires = Ires - (upinteger-FracValue)*V[Next->index];
                Tres = Tres - (upinteger-FracValue);
                LpObj = LpObj + (upinteger-FracValue)*R[Next->index];
        }
```

```
        else{  // in this case, ires > 0, but tres = 0
                Prev = GetPrevDecrX(Next, lbound, value);
                if (Prev != NULL){
                        Ires = Ires - (upinteger-FracValue)*(V[Next->index]-V[Prev->index]);
                        LpObj = LpObj + (upinteger-FracValue)*(R[Next->index]-R[Prev-
>index]);
                        value[Prev->index] = value[Prev->index] - (upinteger - FracValue);
                }
                else{
                        Ires = Ires - (upinteger-FracValue)*V[Next->index];
                        LpObj = LpObj + (upinteger-FracValue)*R[Next->index];
                        Tres = Tres - (upinteger - FracValue);
                        FracValue = -1;
                }
        }
        value[Next->index] = upinteger;
        Next = Next->left;

        *lpobj = LpObj;
        *ires = Ires;
        *tres = Tres;
        *fracvalue = FracValue;
        *Current = Next;
}


void  kozalg()
{
```

```
        int  ii;
        double  LpObj, Ires, Tres, FracValue, UBound[N+1], LBound[N+1], Value[N+1];
        double  LpObj1, Ires1, Tres1, FracValue1, UBound1[N+1], LBound1[N+1],
Value1[N+1];
        VarNode *Current, *Current1, *Temp;
        TreeNode *BestNode;

        make_list();  // arrange variables in a list in non-increasing order of their ratios

        Current = VarList;  // Current now points to first var in the list
        LpObj = 0;
        Ires = I;
        Tres = T;
        FracValue = 0;
        UBound[0] = I; LBound[0] = 0; Value[0] = 0;  // for var R
        for (ii = 1; ii <= N; ii++){            // for vars x
                UBound[ii] = min(T, I/V[ii]);
                LBound[ii] = 0;
                Value[ii] = 0;
        }

        GoRight(1,&LpObj, &Ires, &Tres, &FracValue, UBound, LBound, Value, &Current);
        treeindex = 1;
        InsertTreeNode(treeindex, LpObj, Ires, Tres, FracValue, UBound, LBound, Value,
Current);

        BestNode = TreeList;
```

```
while (BestNode != NULL &&
        (ceil(BestNode->FracVal) - BestNode->FracVal > 0.00000001 &&
         BestNode->FracVal - floor(BestNode->FracVal) > 0.00000001)){
            LpObj1 = BestNode->LpObj; Ires1 = BestNode->Ires; Tres1 =
BestNode->Tres; FracValue1 = BestNode->FracVal; Current1 = BestNode->Current;
            for (ii = 0; ii <= N; ii++){
                UBound1[ii] = BestNode->UBound[ii];
                LBound1[ii] = BestNode->LBound[ii];
                Value1[ii]  = BestNode->Value[ii];
            }
            UndoLeft(&LpObj1, &Ires1, &Tres1, FracValue1, UBound1,
LBound1, Value1, &Current1);
            GoRight(treeindex+1, &LpObj1, &Ires1, &Tres1,
&FracValue1, UBound1, LBound1, Value1, &Current1);
            if (FracValue1 != -1)
                InsertTreeNode(treeindex+1, LpObj1, Ires1, Tres1,
FracValue1, UBound1, LBound1, Value1, Current1);
            LpObj1 = BestNode->LpObj; Ires1 = BestNode->Ires; Tres1 =
BestNode->Tres; FracValue1 = BestNode->FracVal; Current1 = BestNode->Current;
            for (ii = 0; ii <= N; ii++)
                Value1[ii]  = BestNode->Value[ii];
            UBound1[Current1->index] = BestNode->UBound[Current1-
>index];
            FinishRight(&LpObj1, &Ires1, &Tres1, &FracValue1,
UBound1, LBound1, Value1, &Current1);
            GoLeft(treeindex + 2,&LpObj1, &Ires1, &Tres1, &FracValue1,
UBound1, LBound1, Value1, &Current1);
            if (FracValue1 != -1)
                InsertTreeNode(treeindex+2, LpObj1, Ires1, Tres1,
FracValue1, UBound1, LBound1, Value1, Current1);
            DeleteTreeNode(&BestNode);
```

```
                                BestNode = TreeList;
                                treeindex += 2;
        }

        objective_value(BestNode);

        print_results(BestNode);
}


int main()
{
        int ii, numcols, type,i,j;
        double objval, *sol, start1, stop1, start2, stop2;
        long double duration1, duration2;

        insert_data();

        //-------CPLEX solution----------
        start2 = clock();
        if (init_problem() == false) exit;
        if (AddDecVars() == false) exit;
        if (ResourceConstraint() == false) exit;
        if (MultipleChoiceConstraint() == false) exit;

        type = 1;
        if (PrintProblem(Env,Lp,"phaseout.lp") == false) exit;
```

```
        if (type == 0)  // if continuous
          if (LPOpt(Env, Lp) == false) exit;
        if (type == 1)  // if integer
    if (MipOpt(Env, Lp) == false) exit;
        stop2 = clock();

        //--------Kozalg solution-------
        start1 = clock();
        kozalg();
        stop1 = clock();

        free_list();
        if (free_problem() == false) exit;

        duration2 = (long double) (stop2-start2)/CLOCKS_PER_SEC;
        duration1 = (long double) ((stop1-start1)/CLOCKS_PER_SEC);

        printf("\nExecution time of kozalg = %Lf seconds\n", duration1);
        printf("Execution time of cplex = %Lf seconds\n", duration2);

        printf("Press return to continue...\n");
  getchar();

        return 0;

}
```

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΒΙΒΛΙΟΘΗΚΗ
004000125719