Υλοποίηση αναθεωρημένου αλγορίθμου Simplex

Για το γενικό γραμμικό πρόβλημα

Αμπατζόγλου Απόστολος

Διπλωματική Εργασία Μεταπτυχιακού
Εφαρμοσμένης Πληροφορικης

Κατεύθυνση: "Συστήματα Υπολογιστών"

Επιβλέπων Καθηγητής: Σαμαράς Νικόλαος, Λέκτορας

Εξεταστής : Παπαρίζος Κωνσταντίνος, Καθηγητής

Πανεπιστήμιο Μακεδονίας
Θεσσαλονίκη
Ιούνιος 2005

ΠΕΡΙΛΗΨΗ

Η εργασία αυτή πραγματεύεται την επίλυση του γενικού γραμμικού προβλήματος, δηλαδή την μεγιστοποίηση (ελαχιστοποίηση) μιας συνάρτησης της οποίας οι μεταβλητές υπόκεινται σε έναν αριθμό περιορισμών. Για την επίλυση του γραμμικού προβλήματος χρησιμοποιήθηκε ο αναθεωρημένος αλγόριθμος ***Simplex***, καθώς και βοηθητικές προλυτικές διαδικασίες και κλιμάκωση.

Για την αξιολόγηση της υλοποίησης εκτελέστηκε μια εκτεταμένη υπολογιστική μελέτη από την οποία εξήχθησαν χρήσιμα συμπεράσματα για τον αλγόριθμο καθώς και για δύο μεθόδους αντιστροφής μήτρας. Οι δυο μέθοδοι που χρησιμοποιήθηκαν για την αντιστροφή της βάσης (basis) είναι i) η αντιστροφή με χρήση Eta μητρών (***Eta-matrix***) και ii) η αντιστροφή με χρήση ενός εξωτερικού γινομένου και μιας πρόσθεσης μητρών (***Rank One inversion***).

Η γλώσσα προγραμματισμού που χρησιμοποιήθηκε είναι η C++ και συγκεκριμένα το πακέτο Borland C++ Builder 5. Η παρούσα Διπλωματική χωρίζεται σε πέντε ενότητες. Η πρώτη κάνει αναφορά σε κάποιες εισαγωγικές έννοιες και στην δημιουργία των εκτελέσιμων αρχείων ***Metroprograms.exe*** και ***Random.exe*** τα οποία διαβάζουν ήδη υπάρχοντα και δημιουργούν τυχαία σύνολα δεδομένων αντίστοιχα, για επίλυση με τον αλγόριθμο. Στο δεύτερο κεφάλαιο γίνεται εκτενής αναφορά στις ***προλυτικές διαδικασίες*** (presolve) ενώ στο τρίτο αναλύεται ο κυρίως αλγόριθμος καθώς και οι παραλλαγές που χρησιμοποιούνται για ένα σημαντικό κομμάτι του, την ***αντιστροφή μήτρας*** (matrix inversion). Επιπλέον, στα δύο αυτά κεφάλαια περιγράφεται η διαδικασία δημιουργίας του εκτελέσιμου ***Simplex.exe*** το οποίο αποτελεί το πρόγραμμα που θα χρησιμοποιηθεί για την υπολογιστική μελέτη. Τα αποτελέσματα της οποίας συνοδευόμενα από συμπεράσματα και προτάσεις για περαιτέρω έρευνα αποτελούν τα δύο τελευταία κεφάλαια.

ABSTRACT

This paper deals with solving the general linear problem, that is maximizing (minimizing) a function's value. The function's variables submit to a certain number of constraints. The solution of the linear problem is achieved by using the revised ***Simplex*** algorithm, additional presolve operations and scaling.

In order to evaluate the current implementation, a computational survey was performed. From this research there were extracted many useful conclusions about the algorithm and the two basis inversion methods examined. The two methods used for basis inversion were i) eta matrix inversion and ii) rank one inversion.

The programming language used is C++ and particularly the package Borland C++ Builder 5. This paper is divided in five sections. The first refers to some introductory terms and the creation of executable files ***Metroprograms.exe*** and ***Random.exe*** which read existing and create random data sets respectively, which are used for testing the algorithm. In the second chapter there is an advanced analysis of ***presolve operations***, in the third the main algorithm is being examined and the variations used for one of it's most important parts, the ***matrix inversion***. Additionally, in this two chapters the procedure for creating ***Simplex.exe*** is being closely observed. This file is the main program that is used for the computational survey. It's results, conclusions and propositions for future research compose the final two chapters of the paper.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1
ΕΙΣΑΓΩΓΙΚΕΣ ΕΝΝΟΙΕΣ

Το θέμα του γραμμικού προγραμματισμού μπορεί να χωριστεί σε τρεις διαφορετικές αλλά όχι εύκολα διακριτές μεταξύ τους ς: τη θεωρητική, την υπολογιστική και την εφαρμοσμένη. Τα μαθηματικά των γραμμικών ανισοτήτων αναπτύχθηκαν και συνεχίστηκαν από τις υπολογιστικές πλευρές της μεθόδου απαλοιφής για την επίλυση γραμμικών εξισώσεων. Για την επίλυση του προβλήματος του γενικού γραμμικού προγραμματισμού, αναπτύχθηκαν οι μαθηματικές ιδιότητες. Έπειτα, σε μια προσπάθεια να εξηγηθούν πλήρως οι θεμελιώδεις αρχές της μεθόδου Simplex, η ικανότητα να παραχθούν ακραίες λύσεις φάνηκε να είναι μια απλή παραλλαγή της τεχνικής της απαλοιφής των Gauss και Jordan. Σε μια σειρά διαλέξεων παρουσιάστηκαν τα θεωρητικά και τα υπολογιστικά στοιχεία της σύνθετης μεθόδου του G. B. Dantzig [11, 12]. Μια συζήτηση σχετικά με τη δυϊκή υπόσταση των προβλημάτων του γραμμικού προγραμματισμού ακολουθήθηκε από διαλέξεις πάνω στο σχηματισμό συγκεκριμένων ενδεικτικών εφαρμογών (Saul [5]).

Η έρευνα για το βέλτιστο, το μέγιστο, το ελάχιστο ή γενικώς για τις καλύτερες λύσεις σε μια ποικιλία προβλημάτων προκαλούσε το ενδιαφέρον του ανθρώπου από τα αρχαία χρόνια. Ο Ευκλείδης στο βιβλίο ΙΙΙ ασχολήθηκε με την εύρεση της μέγιστα και της ελάχιστα ευθείας γραμμής που μπορεί να σχεδιαστεί από ένα σημείο προς την περιφέρεια ενός κύκλου και στο βιβλίο IV περιέγραψε τον τρόπο εύρεσης ενός παραλληλογράμμου μέγιστης περιοχής με δεδομένη περίμετρο. Ωστόσο, η αυστηρή προσέγγιση σε αυτά τα πιο εξεζητημένα προβλήματα έπρεπε να περιμένουν έως ότου οι μεγάλοι μαθηματικοί του 17ου και 18ου αιώνα αναπτύξουν τις ισχυρές μεθόδους των λογισμών και των λογισμών μεταβολών. Με τις τεχνικές αυτές μπορούμε να βρούμε τις μέγιστες και τις ελάχιστες λύσεις για τη διεύρυνση ενός πεδίου βελτιστοποιημένων προβλημάτων. Αυτές και άλλες μαθηματικές βελτιστοποιημένες διαδικασίες αναφέρονταν κυρίως σε προσπάθειες για την επίλυση των προβλημάτων με τρόπους γεωμετρικής, φυσικής και δυναμικής φύσεως. Τέτοια προβλήματα όπως η αναζήτηση ελάχιστων καμπυλών της περιστροφής και της καμπύλης της ταχύτερης κατάβασης λύνονται με αυτές τις κλασσικές βελτιστοποιημένες μεθόδους (Saul [5]).

Πρόσφατα, προήλθε μια νέα τάξη βελτιστοποιημένων προβλημάτων από τις σύνθετες οργανωτικές δομές που διαποτίζουν τη σύγχρονη κοινωνία. Στην εργασία αυτή θα ασχοληθούμε με τέτοια θέματα ως οι πιο επαρκείς τρόποι για τη οργάνωση μιας οικονομίας ή τη βέλτιστη κατασκευή ενός αεροσκάφους για τη μεγιστοποίηση των πιθανοτήτων μιας χώρας να κερδίσει έναν πόλεμο ή με καθημερινά θέματα όπως η

ανάμειξη των συστατικών ενός λιπάσματος για τις καλύτερες αποδόσεις στην καλλιέργεια με το ελάχιστο κόστος. Έρευνα σχετικά με το σχεδιασμό και τη λύση τέτοιων προβλημάτων οδήγησε στην ανάπτυξη νέων και σημαντικών βελτιστοποιημένων τεχνικών. Το μοντέλο του γραμμικού προγραμματισμού δηλαδή η βελτιστοποίηση του αντικειμένου της γραμμικής λειτουργίας με γραμμικούς περιορισμούς είναι απλό στις μαθηματικές του δομές αλλά ισχυρό στην προσαρμοστικότητα του σε ευρύ φάσμα εφαρμογών (Saul [5]).

Ιστορικά, το γενικό πρόβλημα του γραμμικού προγραμματισμού αναπτύχθηκε πρώτα και εφαρμόσθηκε το 1947 από τους Dantzig [11], Wood και τους συνεργάτες τους στο τμήμα US Air force. Εκείνη την περίοδο η ομάδα αυτή κλήθηκε να ερευνήσει την πιθανότητα εφαρμογής των μαθηματικών τεχνικών σε στρατιωτικά προγράμματα και το σχεδιασμό προβλημάτων. Το ερώτημα αυτό οδήγησε τον Dantzig να προτείνει ότι οι αλληλεξαρτήσεις μεταξύ των δραστηριοτήτων ενός μεγάλου οργανισμού μπορούν να θεωρηθούν ως μοντέλο γραμμικού προγραμματισμού και του βελτιστοποιημένου προγράμματος που ορίζεται από την ελαχιστοποίηση της γραμμικής αντικειμενικής λειτουργίας. Προκειμένου να αναπτυχθούν και να επεκταθούν αυτές οι ιδέες περισσότερο, το Air force οργάνωσε μια ερευνητική ομάδα με την επωνυμία SCOOP (Scientific Computation Of Optimal Problems, επιστημονικός υπολογισμός βέλτιστων προγραμμάτων). Η Air force εκτός από το ότι προγραμμάτισε και χρηματοδότησε προβλήματα σε μια πιο επιστημονική βάση, η κύρια συμβολή του πλάνου SCOOP ήταν η επίσημη ανάπτυξη και εφαρμογή του μοντέλου του γραμμικού προγραμματισμού. Αυτές οι πρώτες εφαρμογές των μεθόδων του γραμμικού προγραμματισμού χωρίστηκαν σε τρεις κύριες κατηγορίες: τις στρατιωτικές εφαρμογές που βασίστηκαν στο πλάνο SCOOP, τις βιομηχανικές οικονομικές εφαρμογές που βασίστηκαν στο μοντέλο εισερχομένων-εξερχόμενων του Leontief και τα προβλήματα που συνδυάζουν τη σχέση μεταξύ του μηδενικού ποσού (zero sum) και των παιχνιδιών δύο ατόμων (two person games). Τα προηγούμενα χρόνια οι τομείς των εφαρμογών αυτών επεκτάθηκαν και αναπτύχθηκαν αλλά η κύρια έμφαση δόθηκε στις εφαρμογές του γραμμικού προγραμματισμού στο γενικό βιομηχανικό τομέα με πολλές εφαρμογές στα κοινωνικά και αστικά πεδία (Saul [5]).

Η αρχική μαθηματική αναφορά στο γενικό πρόβλημα του γραμμικού προγραμματισμού έγινε από τον Dantzig το 1947 [11] μαζί με τη μέθοδο Simplex, μια συστηματική διαδικασία για την επίλυση του προβλήματος. Πριν από αυτό ένας αριθμός προβλημάτων (κάποια άλυτα) αναγνωρίστηκαν ως τύπος προβλημάτων που σχετίζονται με τη βελτιστοποίηση μιας γραμμικής λειτουργίας υπό την προϋπόθεση γραμμικών περιορισμών. Τα πιο σημαντικά παραδείγματα

περιλαμβάνουν το πρόβλημα μεταφοράς που τέθηκε από τον Hitchock (1945) και ανεξάρτητα από τους Koopmans (1947) και το διαιτητικό πρόβλημα του Stigler (1945). Η πρώτη επιτυχής επίλυση ενός προβλήματος γραμμικού προγραμματισμού πραγματοποιήθηκε σε έναν υπολογιστή υψηλής ταχύτητας τον Ιανουάριο του 1952 στην εθνική υπηρεσία των Standard SEAC μηχανών. Από τότε, ο αλγόριθμος Simplex ή οι παραλλαγές αυτής της διαδικασίας κωδικοποιήθηκαν για τους περισσότερους ενδιάμεσους και μεγάλους ηλεκτρονικούς υπολογιστές γενικής χρήσης (Saul [5]).

Ο γραμμικός προγραμματισμός μετατράπηκε σε ένα σημαντικό εργαλείο των σύγχρονων θεωρητικών και εφαρμοσμένων μαθηματικών. Η αξιοσημείωτη εξέλιξη μπορεί να αποδοθεί σε πρωτοποριακές προσπάθειες πολλών ατόμων και ερευνητικών οργανισμών. Στη συνέχεια αυτού του κεφαλαίου θα περιγραφεί το γραμμικό πρόβλημα στην γενική του μορφή, στη συνέχεια θα παρουσιαστεί εκτενώς ο τύπος των προβλημάτων που θα επιλυθούν (μετροπρογράμματα και τυχαία) καθώς και μεθοδολογίες που αφορούν την δημιουργία τέτοιων προβλημάτων (Saul [5]).

1.1 Γενικό γραμμικό πρόβλημα

Ο γραμμικός προγραμματισμός (linear programming) ασχολείται με την επίλυση προβλημάτων βελτιστοποίησης, δηλαδή προσπαθεί να υπολογίσει την μέγιστη ή την ελάχιστη τιμή μιας συνάρτησης της οποίας οι μεταβλητές υπόκεινται σε ένα σύνολο περιορισμών. Ένα τέτοιο πρόβλημα στην γενικότερη του μορφή γράφεται ως εξής:

$$\begin{aligned} \min \quad & z = c^T x \\ \text{μ.π} \quad & Ax \otimes b \\ & x \geq 0 \end{aligned} \qquad \text{(Γ.Π. 1)}$$

όπου A πίνακας πραγματικών αριθμών mxn, c και b πίνακες πραγματικών αριθμών με διαστάσεις 1xn και mx1 αντίστοιχα. Το σύμβολο $\otimes$ υποδηλώνει ότι σε κάθε περιορισμό μπορεί να εμφανίζεται ανισότητα οποιουδήποτε τύπου ή ισότητα (=, ≥, ≤). Η μορφή αυτή ονομάζεται κανονική (canonical).

Για την επίλυση των προβλημάτων αυτών έχουν εφαρμοστεί διάφοροι αλγόριθμοί, οι οποίοι χωρίζονται σε τρεις κύριες κατηγορίες, τους αλγορίθμους εσωτερικών σημείων (interior point methods), τους αλγορίθμους εξωτερικών σημείων (exterior point methods) και τους αλγορίθμους τύπου Simplex. Οι αλγόριθμοι εσωτερικών σημείων εμφανίστηκαν το 1978 από τον Karmarkar, ενώ οι τύπου Simplex είναι κατά πολύ προγενέστεροι τους αφού ο κλασικός αλγόριθμος simplex εμφανίστηκε από τον Dantzig [11] το 1947. Η βασική διαφορά ανάμεσα

στους IMP και στους EPM έγκειται στον τρόπο που επιλέγουν κάποιο αρχικό σημείο με σκοπό να προσεγγίσουν την βέλτιστη λύση. Οι αλγόριθμοι εσωτερικών σημείων (IMP) ξεκινούν από ένα σημείο το οποίο ικανοποιεί όλους τους περιορισμούς ως αυστηρές ανισότητες και στη συνέχεια προσπαθούν να βελτιώσουν την τιμή της αντικειμενικής συνάρτησης, ενώ οι αλγόριθμοι εξωτερικών σημείων ξεκινούν από ένα βασικό εφικτό σημείο και προσεγγίζουν την βέλτιστη λύση μετακινούμενοι και στο εξωτερικό της εφικτής περιοχής.

Τα γραμμικά προβλήματα χαρακτηρίζονται ως μικρά, μεσαία και μεγάλα ανάλογα με το μέγεθος τους. Αυτό δεν χαρακτηρίζεται μόνο από το πλήθος των περιορισμών και των μεταβλητών που συμμετέχουν σε αυτό αλλά και από την πυκνότητα τους, δηλαδή το πόσες μεταβλητές συμμετέχουν σε κάθε περιορισμό. Στην περίπτωση που το γραμμικό πρόβλημα είναι εκφρασμένο σε μορφή μητρών εξαρτάται από το πλήθος των γραμμών, των στηλών και των μη μηδενικών στοιχείων της μήτρας A. Τα μεγάλα γραμμικά προβλήματα (large scale problems), επειδή ο χρόνος επίλυσης τους είναι πολύ μεγάλος διαδραματίζουν σημαντικό ρόλο στη βελτίωση των αλγορίθμων και στην ανάπτυξη νέων. Ως μέτρο σύγκρισης δύο αλγορίθμων χρησιμοποιείται το πλήθος των επαναλήψεων που εκτελεί ο αλγόριθμος μέχρι να φτάσει στη λύση ενός προβλήματος καθώς και ο χρόνος που χρειάζεται για να τις πραγματοποιήσει. Παρ' ότι φαινομενικά ο χρόνος είναι ανάλογος με τον αριθμό των επαναλήψεων, όπως θα αποδειχθεί στη συνέχεια (κεφάλαιο 3) ο ισχυρισμός αυτός δεν είναι πάντα αληθής με αποτέλεσμα να πρέπει να εξετάζονται σαν διαφορετικά μέτρα σύγκρισης. Τέλος, πρέπει να αναφερθεί ότι για την σύγκριση δυο αλγορίθμων τα υπολογιστικά συστήματα που χρησιμοποιούνται πρέπει να έχουν την ίδια υπολογιστική ισχύ (ταχύτητα επεξεργαστή) και ίδια ποσότητα και ταχύτητα κύριας μνήμης.

Τα περισσότερα πραγματικά γραμμικά προβλήματα που επιλύονται δεν είναι πολύ μεγάλα, ο McCall [16] ανέλυσε 16 πραγματικά προβλήματα των οποίων ο μέσος όρος διαστάσεων ήταν 1800 x 2603 και μόνο ένα από αυτά είχε 8965 περιορισμούς και ένα άλλο 7907 μεταβλητές.

Παράδειγμα

«Μια εταιρία κατασκευής σιδήρου πρέπει να αποφασίσει πως θα χρησιμοποιήσει τον χρόνο της επόμενης εβδομάδας σε κάποιο μηχάνημα. Το μηχάνημα αυτό παράγει προϊόντα τύπου Α και προϊόντα τύπου Β, το Α παράγεται με ρυθμό 200 τόνοι ανά ώρα, ενώ το Β με ρυθμό 140 τόνοι ανά ώρα. Επιπλέον, το κέρδος από την πώληση των δύο προϊόντων είναι διαφορετικό, προϊόν Α €25 ανά τόνο ενώ για το Β €30 ανά τόνο. Η μέγιστη ποσότητα που μπορεί να παραχθεί από κάθε προϊόν είναι: προϊόν Α 6000 τόνοι και Β 4000 τόνοι. Λαμβάνοντας υπ' όψιν ότι υπάρχουν 40 ώρες στην επόμενη βδομάδα, σχεδιάσται γραμμικό πρόβλημα το οποίο να αποφασίζει τι ποσότητα από κάθε προϊόν θα παραχθεί.» Έστω ότι η

μηχανή χρησιμοποιείται x1 ώρες για παραγωγή Α και x2 ώρες για παραγωγή Β, τότε:

$$\begin{aligned} \max \quad & (25*200)x_1+(140*30)x_2 \\ \text{μ.π.} \quad & (200*x_1)\leq 6000 \\ & (140*x_2)\leq 4000 \\ & x_1+x_2<40 \end{aligned}$$

Ισοδύναμα:

$$\begin{aligned} \max \quad & 5000x_1+4200x_2 \\ \text{μ.π.} \quad & x_1\leq 30 \\ & x_2\leq 28.57 \\ & x_1+x_2<40 \end{aligned}$$

1.2 Αραιές μήτρες

Στις περισσότερες περιπτώσεις τα γραμμικά προβλήματα έχουν πολύ μικρή πυκνότητα, μερικές φορές μικρότερη και από 1%, και πολύ μεγάλη διάσταση (χιλιάδες γραμμές επί χιλιάδες στήλες). Το γεγονός αυτό κάνει επιτακτική την ανάγκη για χρήση αποθήκευσης διαφορετικής από την πλήρη σε έναν πίνακα διαστάσεων m x n.

Μια τέτοια επιλογή κάνει τα προγράμματα ταχύτερα και δίνει την δυνατότητα επίλυσης μεγαλύτερων προβλημάτων. Λαμβάνοντας υπ' όψιν ότι ένα από τα μεγαλύτερα προβλήματα που θα επιλυθούν έχει μέγεθος 9649 γραμμές επί 72447 στήλες θα απαιτούταν να τοποθετηθούν στην κεντρική μνήμη 6.99e+8 αριθμοί, οι οποίοι αν αποθηκεύονταν σαν long double (64bit) θα χρειαζόντουσαν 5.208 GByte RAM μέγεθος το οποίο είναι απαγορευτικό για οποιονδήποτε υπολογιστή. Η χρήση μιας τέτοιας αποθήκευσης θα μπορούσε να δώσει αποτελέσματα με την χρήση της εικονικής μνήμης (virtual memory) των windows αλλά το χρονικό αποτέλεσμα θα ήταν πολύ κακό επειδή τα δεδομένα θα διαβάζονται από τον αργό σκληρό δίσκο και όχι από την σαφώς ταχύτερη κεντρική μνήμη.

Το μειονέκτημα με την αποθήκευση αραιών μητρών είναι ότι απαιτούν πολυπλοκότερο προγραμματισμό για την εκτέλεση πράξεων μεταξύ πινάκων ακόμα και για την απλή προσπέλαση τους. Το γεγονός όμως αυτό δίνει την δυνατότητα στον προγραμματιστή να παραβλέψει πράξεις μεταξύ μηδενικών στοιχείων, με αποτέλεσμα την πολύ ταχύτερη εκτέλεση της εφαρμογής, αν η ιδιότητα αυτή χρησιμοποιηθεί σωστά.

1.2.1 Τεχνολογίες αραιών μητρών

Ένας από τους πιο απλούς τρόπους αποθήκευσης αραιών μητρών είναι αποθήκευση δύο διανυσμάτων δεικτών και ενός διανύσματος τιμών. Στο πρώτο διάνυσμα δεικτών τοποθετούνται οι δείκτες των γραμμών, στο

δεύτερο οι δείκτες των στηλών, ενώ στο διάνυσμα τιμών οι μη μηδενικές τιμές.

Παράδειγμα
Έστω ο πίνακας

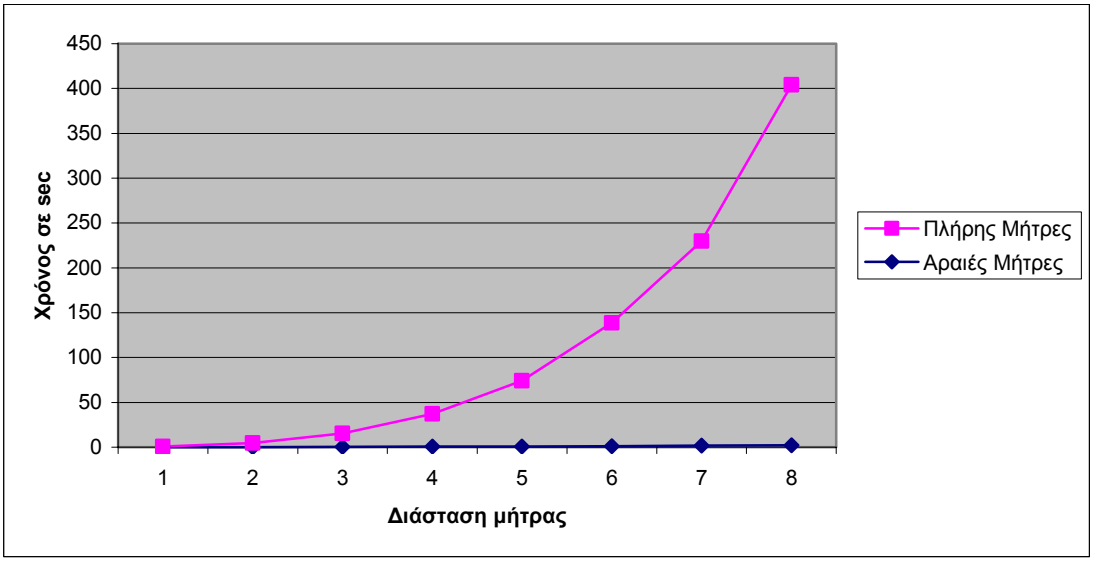
$$A = \begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 \\ 0 & -3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 \end{matrix}$$

αποθηκεύεται στα εξής διανύσματα:

Διάνυσμα Γραμμών = {2 3 4}
Διάνυσμα Στηλών = {1 2 5}
Διάνυσμα Τιμών = {4 -3 5}

Στο παράδειγμα αυτό, ο πίνακας με πυκνότητα 15% αποθηκεύεται με εξοικονόμηση χώρου 55%. Η χρησιμότητα της αποθήκευσης αραιών μητρών γίνεται πιο εμφανής σε μεγαλύτερους πίνακες. Ο Σαμαράς [8] σε μελέτη για τις αραιές μήτρες με την χρήση Matlab παρατήρησε ότι ένας πίνακας 1000x1000 με πυκνότητα 10% καταλαμβάνει μνήμη 1.12 MByte αντί για 7.81 MByte που θα καταλάμβανε η πλήρης αποθήκευση. Επιπλέον, με την χρήση αραιών μητρών ο πολλαπλασιασμός δυο μητρών γίνεται εξαιρετικά πιο γρήγορα και η διαφορά τους μεγαλώνει με την αύξηση των διαστάσεων της μήτρας.

Α/Α	**Μέγεθος**	**Αραιές Μήτρες**	**Πλήρης Μήτρες**
1	500 x 500	0.06	0.6
2	1000 x 1000	0.11	4.45
3	1500 x 1500	0.33	14.94
4	2000 x 2000	0.55	36.58
5	2500 x 2500	0.88	73.22
6	3000 x 3000	1.20	137.31
7	3500 x 3500	1.71	228.10
8	4000 x 4000	2.20	401.78



Εκτός, από την τεχνολογία που αναφέρθηκε, πολύ συχνά χρησιμοποιείται η CSC (Compressed Sparse Column) αποθήκευση. Σε αυτήν αποθηκεύονται και πάλι τρία διανύσματα (Anz, JA, IA) που όμως αναπαριστούν διαφορετικά δεδομένα. Το Anz περιλαμβάνει τις πραγματικές τιμές a_{ij}, i, j=1, 2, …, n της μήτρας A με nz μη μηδενικά στοιχεία. Η αποθήκευση των πραγματικών τιμών a_{ij} γίνεται «σαρώνοντας» τις στήλες 1 έως m. Το πλήθος των στοιχείων της A είναι nz. Το διάνυσμα JA είναι μια λίστα με ακέραιες τιμές η οποία περιλαμβάνει τους δείκτες των γραμμών των μη-μηδενικών στοιχείων a_{ij} με τη σειρά που αποθηκεύονται στη λίστα Anz. Το πλήθος των στοιχείων της λίστας JA είναι nz. Τέλος η λίστα IA περιέχει ακέραιες τιμές η οποία αναπαριστούν δείκτες οι οποίοι προσδιορίζουν την αρχή κάθε στήλης στις λίστες Anz και JA. Το πλήθος των στοιχείων της λίστας IA είναι μικρότερο ή ίσο του n. Στον πίνακα A που αναφέρθηκε παραπάνω η CSC αποθήκευση είναι η εξής:

$$Anz\_A = \{4\ -3\ 5\}$$
$$JA\_A = \{2\ 3\ 4\}$$
$$IA\_A = \{1\ 2\ 3\ 3\ 4\}$$

1.2.2 Υλοποίηση αποθήκευσης στην εφαρμογή

Στην εφαρμογή που αναπτύχθηκε χρησιμοποιήθηκε κυρίως η αποθήκευση CSC. Για την ακρίβεια, χρησιμοποιήθηκε στο διάβασμα των δεδομένων από αρχείο, στη διαδικασία κλιμάκωσης (scaling) και στις προλυτικές διαδικασίες (presolve). Στον αλγόριθμο και στην δημιουργία τυχαίων προβλημάτων χρησιμοποιήθηκε πλήρης αποθήκευση των μητρών.

Όπως έγινε εμφανές από την παράγραφο 1.2.1, για την αποθήκευση μίας μήτρας διατηρούνταν στην μνήμη τρία διανύσματα, αυτά στον κώδικα ονοματίζονται ως Anz_*ΟνομαΜήτρας*, JA_*ΟνομαΜήτρας* και IA_*ΟνομαΜήτρας*. Η δομή δεδομένων που χρησιμοποιήθηκε δεν ήταν απλός pointer της C++ ώστε να μην χρειάζεται δέσμευση χώρου στην αρχή της εκτέλεσης, αλλά μια συνδεδεμένη λίστα η οποία έκανε memory allocation για κάθε νέο κόμβο που εισέρχονταν σε αυτή. Η λίστα δημιουργήθηκε υπό την μορφή template ώστε να μπορούν να δημιουργούνται στιγμιότυπα της με οποιονδήποτε τύπο δεδομένων (π.χ. long double και integer). Η template κλάση περιείχε και κάποιες βασικές συναρτήσεις για την προσπέλαση και τροποποίηση της, όπως addTail(), addHead(), getNth(int), print() κ.α. Αυτός ο τρόπος αποθήκευσης ήταν πολύ αποδοτικός σε συναρτήσεις που απαιτούσαν προσπέλαση του πίνακα κατά στήλες αλλά αρκετά αργός όταν απαιτούταν προσπέλαση κατά γραμμές.

Το αρχείο του template περιέχει την δήλωση της κλάσης Linkedlist όπως φαίνεται παρακάτω:

```
template <class T>
class LinkedList {
    Node<T>* head;
    Node<T>* tail;
    int length;
  public:
    LinkedList() {head=NULL; tail=NULL; length=0;}
    void setList(LinkedList<T>*);
    void addTail (T antikeimeno);
    void removeHead();
    void removeTail();
    bool isEmpty() {return head==NULL;}
    void print();
    void rotate();
    T removeNth(int);
    T getNth(int);
    void setNth(int,T);
    void setNth1(int,T);
    int getLength(){return length;}
    int find(T antikeimeno);
}

template <class T>
class Node {
    T data;
    Node<T>* next;
  public:
    Node() {next=0;}
```

```
    T getData() {return data;}
    void print() {cout<<data<<endl;}
    Node<T>* getNext() {return next;}
    friend LinkedList<T>;
};
```

Ενώ, η δήλωση για παράδειγμα της μήτρας A γίνεται ως εξής:

```
LinkedList<long double>* Anz_A=new LinkedList<long
double>;
LinkedList<long double>* JA_A=new LinkedList<long
double>;
LinkedList<long double>* IA_A=new LinkedList<long
double>:
```

Για το γέμισμα της λίστας με στοιχεία και την προσπέλαση της χρησιμοποιούνται οι συναρτήσεις της κλάσης, η μελέτη των οποίων θεωρείται πλεονασμός.

1.3 Μετροπρογράμματα

Κατά την κατασκευή νέων αλγορίθμων είναι απαραίτητο να γίνει μια υπολογιστική μελέτη που περιλαμβάνει τη σύγκριση του με παλαιότερους. Η σύγκριση αυτή όπως αναφέρθηκε και προηγούμενα γίνεται με βάση δύο κριτήρια, τον αριθμό επαναλήψεων και τον χρόνο CPU. Όπως όμως σε κάθε συγκριτική μελέτη πρέπει οι δύο αλγόριθμοι να εκτελούνται στα ίδια δεδομένα (test sets). Τα προβλήματα που χρησιμοποιούνται για σύγκριση αλγορίθμων γραμμικού προγραμματισμού ονομάζονται μετροπρογράμματα (benchmark).

Τα προγράμματα αυτά έχουν μια συγκεκριμένη μορφή αποθήκευσης η οποία ονομάζεται mps. Η μορφή αυτή είναι παλιό format με συνέπεια να είναι βασισμένη στην χρήση διάτρητων καρτών. Έτσι, τα πεδία δεν διαχωρίζονται με κάποιο διαχωριστικό (separator) αλλά ξεκινάνε σε συγκεκριμένες στήλες κάθε γραμμής. Πιο συγκεκριμένα, τα πεδία ξεκινούν στις στήλες 2, 5, 15, 25 και 40. Επιπλέον, στο αρχείο υπάρχουν συγκεκριμένοι τομείς που προσδιορίζουν ποια μήτρα αναπαριστούν οι εγγραφές. Οι τομείς (section) διαχωρίζονται από κάρτες-κεφαλίδες (header cards) οι οποίες βρίσκονται στην πρώτη στήλη της γραμμής (Gay[14]).

Τα τμήματα που υπάρχουν σε ένα mps αρχείο είναι τα εξής με σειρά εμφάνισης:

1. NAME
2. ROWS

3. COLUMNS
4. RHS
5. RANGES (προαιρετικό)
6. BOUNDS (προαιρετικό)
7. ENDATA

Επιπλέον, ένα mps αρχείο μπορεί να έχει γραμμές με σχόλια, οι οποίες μπορεί να υπάρχουν οπουδήποτε μέσα σε αυτό και ξεχωρίζουν από τις γραμμές δεδομένων από το ότι ξεκινούν με το '*'. Κάθε γραμμή δεδομένων ακολουθεί το παρακάτω fixed-format.

	Πεδίο 1	**Πεδίο 2**	**Πεδίο 3**	**Πεδίο 4**	**Πεδίο 5**	**Πεδίο 6**
Στήλες	2-3	5-12	15-22	25-36	40-47	50-61
Περιεχόμενο	Δείκτης	Όνομα 1	Όνομα 2	Τιμή 1	Όνομα 3	Τιμή 2

Στο τμήμα NAME εμφανίζεται το όνομα του προβλήματος στο πεδίο 4. Στο τμήμα ROWS εμφανίζονται οι τύποι των περιορισμών (ανισοτικοί, ισοτικοί). Στο πεδίο 2 γράφεται το όνομα του περιορισμού και στο πεδίο 4 ο τύπος σύμφωνα με τον παρακάτω πίνακα:

Τύπος	**Σύμβολο**	**Επεξήγηση**
=	E	Ίσο με
<	L	Μικρότερο ή ίσο με
>	G	Μεγαλύτερο ή ίσο με
Objective	N	Όνομα αντικειμενικής συνάρτησης
Free	N	Χωρίς περιορισμό

Το πρώτο όνομα με τιμή N που εμφανίζεται είναι το όνομα της αντικειμενικής συνάρτησης. Στο τμήμα COLUMNS εμφανίζονται τα περιεχόμενα της μήτρας A. Στο πεδίο 2, αναφέρεται το όνομα της μεταβλητής στα πεδία 3 και 5 αναγράφονται τα ονόματα των περιορισμών στα οποία συμμετέχουν (όπως αυτά δηλώθηκαν στο τμήμα ROWS) και στα πεδία 4 και 6 οι συντελεστές των συγκεκριμένων μεταβλητών στους περιορισμούς. Στο τμήμα RHS καταγράφονται τα δεξιά μέρη των περιορισμών. Στο πεδίο 3 γράφεται το όνομα του περιορισμού (όπως στο ROWS) και στο πεδίο 4 η τιμή του διανύσματος b στη συγκεκριμένη θέση. Το τμήμα RANGES σε περίπτωση που υπάρχει υποδηλώνει ότι κάποιος περιορισμός εκτός από δεξί μέρος έχει και αριστερό (διάνυσμα b1) είναι δηλαδή της μορφής $l \leq A \leq u$. Τέλος, το τμήμα BOUNDS μας δείχνει αν κάποια μεταβλητή έχει κάποιον περιορισμό στην τιμή της, σε περίπτωση που κάποια μεταβλητή δεν εμφανίζεται σε αυτό το τμήμα θεωρούμε ότι ισχύει $x_i \geq 0$ (Gay[14]).

Όλα τα παραπάνω συνοψίζονται στον παρακάτω πίνακα:

α/α πεδίου	1	2	3	4	5	6
Στήλες	2-3	5-12	15-22	25-36	40-47	50-61
	NAME		Όνομα προβλήματος			
	ROWS					
	Είδος περιορισμού	Όνομα περιορισμού				
	COLUMNS					
		Όνομα μεταβλητής	Όνομα περιορισμού	Τιμή	Όνομα περιορισμού	Τιμή
	RHS					
		Όνομα δεξιού μέρους	Όνομα περιορισμού	Τιμή	Όνομα περιορισμού	Τιμή
	RANGES					
		Όνομα περιοχής	Όνομα περιορισμού	Τιμή	Όνομα περιορισμού	Τιμή
	BOUNDS					
	Είδος ορίου	Όνομα ορίου	Όνομα περιορισμού	Τιμή	Όνομα περιορισμού	Τιμή
	ENDATA					

Για την καλύτερη κατανόηση των παραπάνω στη συνέχεια παρουσιάζεται το αρχείο mps ενός γραμμικού προβλήματος.

```
NAME          slp10_10x10_1118250584
ROWS
 G  R1
 G  R2
 G  R3
 G  R4
 G  R5
 G  R6
 G  R7
 G  R8
 G  R9
 G  R10
 N  COST
COLUMNS
    X1        COST      7
    X1        R3        2
    X2        COST      4
    X2        R6        3
```

```
    X3          COST        8
    X3          R3          1
    X3          R8          1
    X4          COST        9
    X5          COST        9
    X5          R5          1
    X5          R7          3
    X6          COST        4
    X7          COST        3
    X7          R5          1
    X8          COST        5
    X8          R4          3
    X9          COST        2
    X10         COST        8
    X10         R7          2
    X10         R9          4
RHS
    RANGE       R2          266
    RANGE       R3          0
    RANGE       R4          18
    RANGE       R5          7
    RANGE       R6          48
    RANGE       R7          37
    RANGE       R8          6
    RANGE       R9          24
ENDATA
```

Το προηγούμενο mps αντιστοιχεί στο παρακάτω γραμμικό πρόβλημα γραμμένο στην κλασική μαθηματική μορφή.

Γραμμικό Πρόβλημα

$MIN\ 7X_1 + 4X_2 + 8X_3 + 9X_4 + 9X_5 + 4X_6 + 3X_7 + 5X_8 + 2X_9 + 8X_{10}$

SUBJECT TO

$R3)\ 2X_1 + X_3 >= 0$

$R4)\ 3X_8 >= 18$

$R5)\ X_5 + X_7 >= 7$

$R6)\ 3X_2 >= 48$

$R7)\ 3X_5 + 2X_{10} >= 37$

$R8)\ X_3 >= 6$

$R9)\ 4X_{10} >= 24$

1.3.1 mps to lp

Τα αρχεία των προγραμμάτων benchmark μπορούν να βρεθούν στο internet στη διεύθυνση www.netlib.org/lp/data. Τα προβλήματα αυτά χωρίζονται σε τρεις κατηγορίες τα βέλτιστα (optimal), τα αδύνατα

(infeasible) και τα Kennington. Από τα προβλήματα αυτά στην εργασία θα μελετηθούν 54 βέλτιστα, 6 αδύνατα και 6 Kennington.

Πρώτο στάδιο για την επίλυση ενός μετροπρογράμματος αποτελεί το διάβασμα του mps αρχείου και η αποθήκευση των δεδομένων στην μνήμη με κάποια από τις μορφές που αναφέρθηκαν στην παράγραφο 1.2.1. Στην υλοποίηση που επιλέχθηκε η διαδικασία αυτή γίνεται σε δύο στάδια και με την χρήση δύο συναρτήσεων. Η πρώτη (mpstolp) διαβάζει τα δεδομένα από το αρχείο mps και τα καταγράφει σε ένα νέο αρχείο με επέκταση lp. Η μορφή του αρχείου lp είναι η εξής:

PROGRAM NAME: program_name
==========
min/max {-1/1}
equin {Anz#IA#JA}
b {Anz#IA#JA}
b1 {Anz#IA#JA}
c^T *{Anz#IA#JA}*
A {Anz#IA#JA#}
Bu {Anz#IA#JA}
Bl {Anz#IA#JA}
Bi {IA}
==========
Οι τομείς 1-9 χωρίζονται με αλλαγή γραμμής
Οι πίνακες αποθηκεύονται σε CSC μορφή
Τα διανύσματα κάθε πίνακα διαχωρίζονται με το σύμβολο '#'

Η δεύτερη συνάρτηση (readToTable), διαβάζει σειριακά το αρχείο lp και με χρήση της συνάρτησης addTail της κλάσης LinkedList γεμίζει τις λίστες στην RAM. Όπως είναι εύκολα κατανοητό το γράψιμο του ενδιάμεσου αρχείου επιβαρύνει τον χρόνο εκτέλεσης του προγράμματος, αλλά μπορεί να αποτελέσει μία εναλλακτική μορφή για εισαγωγή δεδομένων στο πρόγραμμα και να βοηθήσει ουσιαστικά στην διαδικασία αποσφαλμάτωσης του κώδικά. Όπως προκύπτει από πειραματική μελέτη ακόμη και τα πολύ μεγάλα προβλήματα δεν απαιτούν σημαντικό χρόνο για την εκτέλεση αυτής της συνάρτησης, οπότε η χρήση της δεν αντενδείκνυται.

Μεγάλο ενδιαφέρον παρουσιάζει ο τρόπος υλοποίησης της συνάρτησης που διαβάζει το mps αρχείο λαμβάνοντας υπ' όψιν ότι τα αποτελέσματα της είναι πάρα πολύ γρήγορα και δεν είναι απαγορευτικά για κανένα μετροπρόγραμμα. Ενδεικτικό για το γεγονός αυτό είναι ότι ακόμα και τα Kennington προβλήματα διαβάζονται σε χρόνους μικρότερους των 90 δευτερολέπτων χωρίς σπατάλη μνήμης (το cre_b δεσμεύει μόνο 35MByte RAM). Περισσότερες λεπτομέρειες για τους

χρόνους εκτέλεσης της συνάρτησης mps δίνονται από τον παρακάτω πίνακα.

Όνομα	Μέγεθος	Read Time	Write Time	Total Time
25fv47	822 x 1571	0.06	0.11	0.17
ship12l	1152 x 5427	0.36	0.21	0.57
Stocfor2	2158 x 2031	0.13	0.09	0.22
Stocfor3	16676 x 15695	14.901	0.751	15.652
osa-14	2338 x 52460	10.905	3.525	14.43
cre_b	9649 x 72447	73.234	3.255	76.509
cre_d	8927 x 69980	59..595	3.094	62.689

Η ταχύτητα εκτέλεσης του προβλήματος οφείλεται στο ότι το αρχείο διαβάζεται μόνο μία φορά και αποθηκεύεται ολόκληρο στην μνήμη με τρόπο ώστε να γράφεται σειριακά στο δεύτερο αρχείο με μια και πάλι μόνο προσπέλαση. Για να επιτευχθεί αυτό χρησιμοποιούνται τρεις νέες δομές δεδομένων, η κλάση ConstraintVarValue, η κλάση Constraint και η κλάση Var.

Στη συνέχεια δημιουργούνται τρεις συνδεδεμένες λίστες με την χρήση του template LinkedList (παράγραφος 1.2.2), μία τύπου ConstraintVarValue, μια τύπου Var και μία τύπου Constraint. Πιο συγκεκριμένα, κάθε αντικείμενο τύπου Var περιέχει μια συνδεδεμένη λίστα τύπου ConstraintVarValue. Η επιλογή αυτή έγινε επειδή λόγω της CSC αποθήκευσης οι μήτρες θα προσπελαστούν κατά στήλες, δηλαδή κατά μεταβλητές.

```
class Constraint {
    int constId;
    char name[20];
    int equin;
    long double rhs;
    long double lhs;
    bool objective;
  public:
    Constraint() {};
    Constraint(int constId_,char name_[], int
equin_, bool obj_)
    char* getName() {return name;}
    int   getId() {return constId;}
    bool  getObj() {return objective;}
    int   getEquin() {return equin;}
    long double   getRHS() {return rhs;}
    long double   getLHS() {return lhs;}
```

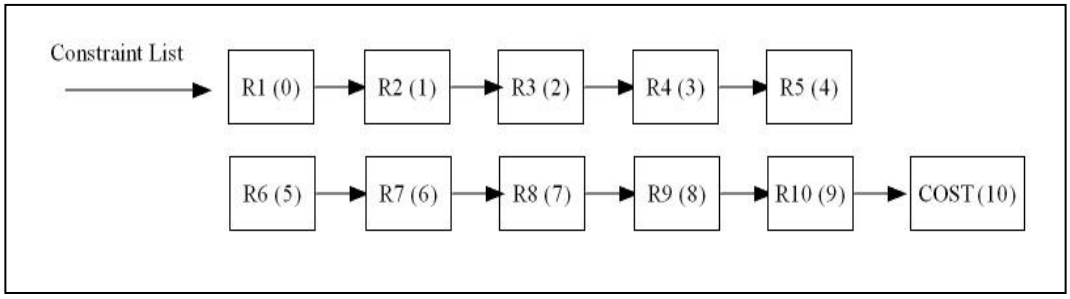
```
    void setRHS(long double value_) {rhs=value_;}
    void setLHS(long double value_)     void
setId(int id_) {constId=id_;}
    void setObj(bool obj_) {objective=obj_;}
    void setEquin(int equin_) {equin=equin_;}
    bool nameEquals(char[]);
};
```

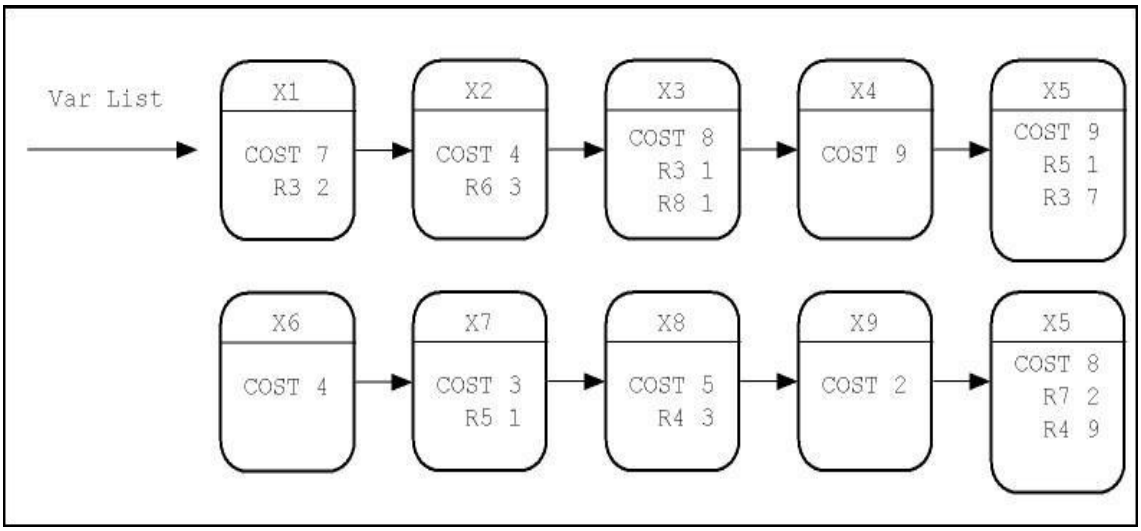
```
class Var {
    char name[20];
    long double rbound;
    long double lbound;
    bool binary;
    LinkedList<constraintVarValue> *constraintApp;
  public:
    Var() {}
    Var(char name_[20], constraintVarValue
constrVar)
    char* getName() {return name;}
    long double getBu() {return rbound;}
    long double getBl() {return lbound;}
    bool getBinary() {return binary;}
    void addTail(constraintVarValue constrVar)
    long double getObjWeight(int);
    void setLeftBound (long double a) {lbound=a;}
    void setRightBound (long double a) {rbound=a;}
    void setBinary (bool a) {binary=a;}
    bool nameEquals(char[]);
    void printValues(int,FILE*);
    void printIds(int,FILE*);
    void printOrders(int,int&);
};
```

```
class constraintVarValue {
    int constId;
    long double value;
  public:
    constraintVarValue() {}
    constraintVarValue(int constrId_, double value_)
{constId=constrId_; value=value_;}
    void setId(int id_) {constId=id_;}
    int getId() {return constId;}
    void setValue(long double value_) {value=value_;}
    long double getValue() {return value;}
};
```

Η συνάρτηση διαβάζει αρχικά το πεδίο ROWS, και δημιουργείται η λίστα Constraint με κόμβους τους περιορισμούς του προβλήματος. Διαβάζοντας το πεδίο COLUMNS ανακτά την πρώτη μεταβλητή, σε περίπτωση που αυτή δεν υπάρχει στη λίστα Var την προσθέτει και επεκτείνει την λίστα ConstraintVarValue με την τιμή που συμμετέχει στον περιορισμό. Σε περίπτωση που η μεταβλητή υπάρχει στην λίστα απλά ανανεώνεται η λίστα ConstraintVarValue.

Μετά από το τέλος αυτής της διαδικασίας για το μετροπρόγραμμα της παραγράφου 1.3 η μνήμη RAM περιέχει τις εξής συνδεδεμένες λίστες:





Τέλος, πολύ σημαντικό για την γρήγορη εκτέλεση του προγράμματος είναι ότι επειδή καμία από τις λίστες δεν ξαναχρησιμοποιείται δεν γίνεται δηλαδή, ανάκτηση των περιεχομένων της με μια while της μορφής:

```
i=0;
while (1) {
    if (list->isEmpty()==true) break;
    data=list->getNth(i);
    i++
}
```

αλλά της μορφής:

```
while (1) {
    if (list->isEmpty()==true) break;
    data=list->getHead();
    list->removeHead();
}
```

Η συνάρτηση getHead έχει σταθερό χρόνο εκτέλεσης, ίσο με αυτό της εκτέλεσης της συνάρτησης getNth(1), ενώ η κλήση της συνάρτησης getNth(n) έχει χρόνο εκτέλεσης ανάλογο του n, αφού γίνεται προσπέλαση n κόμβων. Η αλλαγή αυτή και μόνο προκαλεί σημαντική μείωση του χρόνου εκτέλεσης του προγράμματος για μεγάλα set δεδομένων. Για παράδειγμα στο μετροπρόγραμμα stocfor2 που έχει 2031 μεταβλητές στην δεύτερη while γίνονται 2031 προσπελάσεις κόμβων της λίστας LinkedList<Var>, ενώ με το πρώτο while γίνονται 1 + 2 + 3 + … + 2029 + 2030 + 2031 προσπελάσεις κόμβων της ίδιας λίστας.

1.3.2 write mps

Εξίσου σημαντική με την διαδικασία διαβάσματος ενός mps αρχείου, είναι και η αντίστροφη διαδικασία. Αυτή δηλαδή η οποία από το διάβασμα των διανυσμάτων στην κεντρική μνήμη γράφει στον δίσκο ένα αρχείο σε mps format. Η συνάρτηση αυτή ονομάζεται writeMPS και χρησιμοποιείται για δύο λόγους, πρώτα για την δημιουργία τυχαίων προβλημάτων από τον υπολογιστή και δεύτερον μετά τις προλυτικές διαδικασίες, πριν την επίλυση του αλλαγμένου γραμμικού προβλήματος με τον αλγόριθμο, με σκοπό τον έλεγχο της ορθότητας των μέχρι εκείνο το σημείο ενεργειών.

Η writeMPS είναι πολύ απλή στην υλοποίηση της αφού απλά διαβάζει σειριακά τα τρία διανύσματα (Anz,JA και IA) και γράφει στο fixed-format που επιβάλει η δομή του αρχείου.

```
 fputs(string,out);
  Temp=new LinkedList<long double>;
  Temp1=new LinkedList<long double>;
  while(1) {
     tmp=Anz_B->getNth(1);
     temp=JA_B->getNth(1);
     if (temp==0) break;
     Anz_B->removeHead();
     Temp->addTail(tmp);
     JA_B->removeHead();
     Temp1->addTail(temp);
     strcpy(string,"    RANGE     R");
```

```
      strcat(string,fcvt(temp,0,&dec,&sign));
      if (temp<10) strcat(string,"        ");
         else if (temp<100) strcat(string,"       ");
             else if (temp<1000) strcat(string,"    ");
             else if (temp<10000) strcat(string,"   ");
      strcat(string,fcvt(tmp,0,&dec,&sign));
      strcat(string,"\n");
      fputs(string,out);
   }
   Anz_B->setList(Temp);
   JA_B->setList(Temp1);
```

Τα μόνα σημεία που χρίζουν σχολιασμού στον παραπάνω κώδικα αποτελεί ο έλεγχος για το μέγεθος του ονόματος του περιορισμού, που ορίζει το πλήθος των κενών που θα προστεθούν ώστε η επόμενη στήλη να αρχίζει πάντα σε συγκεκριμένο σημείο και η προσπέλαση των διανυσμάτων (Anz_B και JA_B στο συγκεκριμένο παράδειγμα). Έτσι, όπως και προηγούμενα η προσπέλαση της λίστας γίνεται με χρήση removeHead και getNth(1) αντί για getNth(n). Στο σημείο όμως αυτό επειδή οι λίστες Anz_B και JA_B ξαναχρησιμοποιούνται, χρησιμοποιούμε δύο προσωρινές λίστες Temp και Temp1 στις οποίες προστίθενται οι τιμές που γράφονται στο αρχείο. Στο τέλος της διαδικασίας αντιγράφεται η προσωρινή λίστα στην αρχική η οποία έχει μείνει κενή.

1.4 Γεννήτρια τυχαίων προβλημάτων

Συνήθως στις υπολογιστικές μελέτες αλγορίθμων εκτός από τις δοκιμές με ειδικά προβλήματα (benchmark), τα προγράμματα εκτελούνται και σε τυχαία δεδομένα. Λαμβάνοντας υπ' όψιν ότι η εφαρμογή δέχεται ως είσοδο αρχεία mps, χρειάζεται η δημιουργίας μιας συνάρτησης (readToList) που να δημιουργεί τη μήτρα A και τα διανύσματα b, c και equin. Επιπλέον, η συνάρτηση αυτή θα πρέπει να έχει την δυνατότητα να δημιουργεί αραιά προβλήματα οποιασδήποτε πυκνότητας και μεγέθους, βέλτιστα ή μη.

Από τα παραπάνω ιδιαίτερη μεταχείριση απαιτείται για την δημιουργία αραιών βέλτιστων τυχαίων προβλημάτων. Για να δημιουργηθούν προβλήματα αυτής της μορφής αρχικά γεμίζουν οι πίνακες A και c με τυχαία νούμερα σε συγκεκριμένο εύρος τιμών που δίνεται από τον χρήστη και στη συνέχεια εξασφαλίζεται η βελτιστότητα του γ.π. με ανάθεση κατάλληλων τιμών στο διάνυσμα b.

```
for (i=0;i<rows;i++)
    for (j=0;j<cols;j++)
          A[i][j]=0;
num1=rows*cols*pct/100;
for (temp=0;temp<num1;temp++) {
     aaa=rand()%(rA-lA+1)+lA;
     i=rand()%rows;
     j=rand()%cols;
     if (A[i][j]==0) A[i][j]=aaa;
         else temp--;
}

for (j=0;j<cols;j++) {
     aaa=rand()%(rC-lC+1)+lC;
     C[j]=aaa;
}
```

Ο b γεμίζει σύμφωνα με τον τύπο, b=y*a, με y=k+r*(a/norm(a)).

Παράδειγμα

$$A = \begin{matrix} 0 & 4 & 3 \\ 2 & 3 & 1 \\ 4 & 4 & 1 \end{matrix}$$

τότε,

$$\text{normA} = \begin{matrix} 5 & 3.7 & 5.7 \end{matrix}$$

$$Y = \begin{matrix} 2 & 6 & 5 \\ 5 & 7 & 3 \\ 6 & 6 & 3 \end{matrix} \quad \text{και}$$

$$b^T = \begin{matrix} 39 & 34 & 51 \end{matrix}$$

```
for (i=0;i<rows;i++) {
    aaa=norm(rows,A[i]);
    if (aaa!=0) for (j=0;j<cols;j++)
Y[j]=center+radius*(A[i][j]/aaa);
           else Y[i]=0;
    B[i]=0;
    for (j=0;j<cols;j++) B[i]=B[i]+A[i][j]*Y[j];
}
```

Οι τιμές center και radius δίνονται από τον χρήστη με μοναδική προϋπόθεση center>0 και radius<center, ώστε ο κύκλος που σχηματίζεται με κέντρο center και ακτίνα radius να βρίσκεται αποκλειστικά στο 1ο τεταρτημόριο.

Στο σημείο αυτό τονίζεται το γεγονός ότι όλοι οι περιορισμοί πρέπει να είναι της μορφής ≤. Τα αρχεία που δημιουργούνται ονομάζονται σύμφωνα με τις ιδιότητες τους. Πιο συγκεκριμένα, το πρώτο γράμμα υποδηλώνει το αν είναι πυκνό ή αραιό, στη συνέχεια προσδιορίζεται η πυκνότητα και το μέγεθος του προβλήματος. Παραδείγματος χάριν, ένα αραιό πρόβλημα πυκνότητας 10% με μέγεθος 1000 x 1000 θα ονοματιζόταν slp10_1000x1000.

ΚΕΦΑΛΑΙΟ 2
ΠΡΟΛΥΤΙΚΕΣ ΔΙΑΔΙΚΑΣΙΕΣ

Ένας μεγάλος αριθμός γραμμικών προβλημάτων είναι αδύνατο να επιλυθούν στην αρχική τους μορφή, όπως διαβάζονται δηλαδή από τα αρχεία mps. Το γεγονός αυτό οφείλεται στο ότι συνήθως αυτά τα προβλήματα παράγονται αυτόματα από εργαλεία μοντελοποίησης. Έτσι, απαραίτητη είναι η απλοποίηση των γραμμικών συστημάτων με κάποιες διαδικασίες που ονομάζονται προλυτικές (presolve procedures).

Στο κεφάλαιο αυτό αναλύονται οι σημαντικότερες από αυτές, καθώς και τεχνικές κλιμάκωσης και ανοχές, οι οποίες βοηθούν τον λύτη να φτάσει γρήγορα και με περισσότερη ακρίβεια στη βέλτιστη τιμή της αντικειμενικής συνάρτησης (Forrest, Tomlin [13]).

2.1 Τεχνικές κλιμάκωσης

Ένας από τους πλέον σημαντικούς λόγους που τα προβλήματα benchmark δεν μπορούν να επιλυθούν είναι το μεγάλο τους μέγεθος και η διαφορά τάξης μεγέθους στα περιεχόμενα της μήτρας A. Ο συνδυασμός των δύο αυτών χαρακτηριστικών εμφανίζει κατά την εξέλιξη του αλγορίθμου στοιχεία περιστροφής πολύ κοντά στο μηδέν τα οποία εντείνουν το πρόβλημα, με αποτέλεσμα κάποια στιγμή να εμφανιστεί στοιχείο περιστροφής ίσο με το μηδέν. Λαμβάνοντας υπ' όψιν ότι το στοιχείο περιστροφής είναι παρονομαστής σε μία διαίρεση το γεγονός αυτό προκαλεί run-time errors.

Το πρόβλημα αυτό επιλύεται με τις τεχνικές κλιμάκωσης, με αυτές τα στοιχεία της μήτρας A γίνονται πιο κοντινά σε τάξη μεγέθους με αποτέλεσμα να βελτιώνεται η αριθμητική ακρίβεια και να μειώνεται η αριθμητική προσπάθεια που απαιτείται για την επίλυση του γραμμικού προβλήματος (Σαμαράς [8]). Οι πρώτες τεχνικές κλιμάκωσης αναπτύχθηκαν από τον Tomlin [16] και παραμένουν οι πιο ευρέως χρησιμοποιούμενες. Στην εφαρμογή χρησιμοποιήθηκε η τεχνική της εξισορρόπησης (equilibration technique).

Αρχικά εντοπίζεται το μεγαλύτερο κατ' απόλυτη τιμή στοιχείο της κάθε στήλης της μήτρας A και όλα τα στοιχεία της στήλης διαιρούνται με αυτό. Το αποτέλεσμα αυτής της διαδικασίας είναι σε κάθε στήλη ο μεγαλύτερος κατ' απόλυτη τιμή αριθμός να είναι το ±1. Οι μέγιστες τιμές κάθε στήλης αποθηκεύονται στο διάνυσμα colmulti=1/colmax. Στη συνέχεια η μήτρα σαρώνεται κατά γραμμές. Στις γραμμές όπου δεν υπάρχει ο αριθμός ±1 επαναλαμβάνεται η διαδικασία κλιμάκωσης και το μέγιστο στοιχείο αποθηκεύεται σε ένα διάνυσμα rowmulti=1/rowmax.

Κατά την παραπάνω διαδικασία ο πολλαπλασιασμός με το colmulti επηρεάζει το διάνυσμα κόστους c, ενώ ο πολλαπλασιασμός με το rowmulti το δεξιό μέρος b.

Σ' αυτό το σημείο σημαντικό είναι να τονιστεί ότι το κλιμακωμένο πρόβλημα έχει την ίδια βέλτιστη τιμή στην αντικειμενική συνάρτηση με το αρχικό, αλλά διαφέρουν οι τιμές της λύσης x. Πιο συγκεκριμένα $X_{original} = (colmulti)^T \otimes X_{scaled}$. Σε ορισμένα προβλήματα η χρήση τεχνικών κλιμάκωσης μειώνει τον αριθμό των επαναλήψεων που απαιτούνται για την εύρεση της λύσης, ενώ σε άλλα η λύση τους δεν είναι καν υπολογίσιμη χωρίς αυτές.

Η υλοποίηση του scaling είναι μια σχετικά αργή διαδικασία λόγω της αραιής αποθήκευσης κατά στήλες. Έτσι, η σάρωση του πίνακα κατά γραμμές απαιτεί τόσες σαρώσεις της λίστας Anz_A όσες είναι και το πλήθος των γραμμών όπου δεν βρέθηκε η μονάδα. Για την αντιμετώπιση αυτού του προβλήματος ώστε να γίνεται μόνο μια σάρωση του Anz_A εκτελείται ο παρακάτω αλγόριθμος:

1. Δημιουργία δύο κενών λιστών Anz_rowmulti1 και JA_rowmulti1
2. Σάρωση του Anz_A και του JA_A, αν η τιμή του JA_A δεν υπάρχει στην λίστα JA_rowmulti1 τότε προστίθεται σε αυτήν καθώς και η αντίστοιχη τιμή του Anz_A στον Anz_rowmulti1. Σε περίπτωση που το JA_A υπάρχει στον JA_rowmulti1 στην θέση row, τότε παίρνουμε το στοιχείο row του Anz_rowmulti1 και το συγκρίνουμε με το στοιχείο του Anz_A σε περίπτωση που αυτό είναι μεγαλύτερο το αντικαθιστούμε.
3. Σαρώνουμε τον Anz_rowmulti1 και τον JA_rowmulti1 και όπου δεν υπάρχει η μονάδα, περνάμε στη θέση JA_rowmulti1 του διανύσματος rowmulti την τιμή 1/Anz_rowmulti1.

```
while (1) {
  col=JA_A->getNth(1);
  if (col==0) break;
  pos=JA_rowmulti1->find(col);
  value=Anz_A->getNth(1);
  if (pos==0) {
    Anz_rowmulti1->addTail(fabs(value));
    JA_rowmulti1->addTail(col);
  } else {
        long double k=Anz_rowmulti1->getNth(pos);
        k=fabs(k);
        if (fabs(value)>k)
            Anz_rowmulti1->setNth1(pos,fabs(value));
  }
  JA_A->removeHead();
  Temp1->addTail(col);
  Anz_A->removeHead();
  Temp->addTail(value);
}
Anz_A->setList(Temp);
JA_A->setList(Temp1);
```

```
while(1) {
  value=Anz_rowmulti1->getNth(1);
  col=JA_rowmulti1->getNth(1);
  if (col==0) break;
  Anz_rowmulti1->removeHead();
  JA_rowmulti1->removeHead();
  if (value!=1) {
      Anz_rowmulti->addTail(1/value);
      JA_rowmulti->addTail(col);
  }
}
```

2.2 Preprocessing

Όπως αναφέρθηκε και προηγούμενα, η απλοποίηση ενός γραμμικού προβλήματος κρίνεται επιτακτική σε περίπτωση που αυτό είναι μεγάλης κλίμακας. Το αποτέλεσμα της απλοποίησης είναι ένα ισοδύναμο πρόβλημα με το αρχικό αλλά με μικρότερες διαστάσεις. Για αυτό τον μετασχηματισμό εφαρμόζονται διάφορες ευρετικές μέθοδοι (heuristic methods) και αλγόριθμοι. Η ανάλυση αυτή ονομάζεται προλυτική ανάλυση. Με αυτήν επιτυγχάνεται μείωση του χρόνου επίλυσης του προβλήματος και έτσι είναι δυνατή η επίλυση γραμμικών προβλημάτων τα οποία δεν μπορούν να λυθούν στην αρχική τους μορφή.

Η ελάττωση των διαστάσεων του προβλήματος έχει ως αποτέλεσμα την αύξηση της υπολογιστικής αποτελεσματικότητας του αλγορίθμου, γιατί η πολυπλοκότητα του αλγορίθμου simplex εξαρτάται από την διάσταση του γραμμικού προβλήματος. Η καλυτέρευση των αριθμητικών ιδιοτήτων του προβλήματος έχει ως αποτέλεσμα την μείωση των σφαλμάτων ακρίβειας, ενώ μπορεί να εντοπιστεί αν κάποιο πρόβλημα είναι αδύνατο ή απεριόριστο χωρίς να εκτελεστεί ο αλγόριθμος. Τέλος, με τη χρήση προλυτικών λειτουργιών επισημαίνονται ορισμένες ιδιότητες του προβλήματος όπως ο εκφυλισμός, η κύκλωση και το μέγεθος των δεδομένων του (Σαμαράς [8]).

Σύμφωνα με την μελέτη του Παπαρίζου [7], οι προλυτικές διαδικασίες μπορούν εκτός από το αρχικό πρόβλημα να εκτελεστούν και στο νέο πρόβλημα που κατασκευάζεται. Στη συγκεκριμένη εφαρμογή, τα presolve εκτελέστηκαν τόσες φορές ώστε μετά την ολοκλήρωση ενός πλήρους κύκλου εκτελέσεων να μην προκύψει καμία μείωση στο μέγεθος του προβλήματος, παρ' ότι η διαδικασία αυτή είναι εξαιρετικά χρονοβόρα.

Οι παράγραφοι που ακολουθούν, επεξηγούν τη λειτουργία των προλυτικών διεργασιών. Η ομαδοποίηση τους και η παρουσίαση τους δεν γίνεται σύμφωνα με τη μαθηματική τους υπόσταση, αλλά σύμφωνα με τον

τρόπο που υλοποιήθηκαν σε συναρτήσεις κατά τη διάρκεια του προγραμματισμού.

2.2.1 Εντοπισμός και διαγραφή κενών γραμμών και στηλών

Έστω ο περιορισμός της μορφής,

$$b\_left \le a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n \le b\_right$$

Ο περιορισμός αυτός ονομάζεται κενός αν $a_{ij}=0$ για κάθε j=1,2,…,n και i=1, 2, …, m. Ένας περιορισμός αυτής της μορφής μπορεί να είναι είτε πλεονασματικός, είτε αδύνατος. Στην περίπτωση που ο περιορισμός είναι πλεονασματικός μπορεί να διαγραφεί.

Ο περιορισμός είναι πλεονασματικός στις εξής περιπτώσεις:

- $b\_right \ge 0$, $b\_left = 0$
- $b\_left \le 0$, $b\_right = 0$
- $b\_right = b\_left = 0$

Το πρόβλημα είναι αδύνατο στις εξής περιπτώσεις:

- $b\_left \ge 0$, $b\_right = 0$
- $b\_right \le 0$, $b\_left = 0$
- $b\_right = b\_left \ne 0$

Αντίστοιχα, μια στήλη ονομάζεται κενή αν ισχύει $a_{ij}=0$ για κάθε i = 1, 2, …, m και j = 1, 2, …, n. Σε αυτή την περίπτωση μπορεί είτε η μεταβλητή να είναι πλεονασματική είτε το πρόβλημα να είναι απεριόριστο. Πιο συγκεκριμένα, αν η κενή στήλη είναι η j τότε η μεταβλητή είναι πλεονασματική αν ισχύει $c_j \ge 0$ και μπορεί να διαγραφεί. Σε αντίθετη περίπτωση το πρόβλημα είναι απεριόριστο.

Παράδειγμα

Έστω ένα γραμμικό πρόβλημα με τις μήτρες αποθηκευμένες σε CSC αποθήκευση:

Anz_B = {5 7 -1 5} JA_B = {1 2 3 4}
Anz_C = {-1 2 1 -4 -3} JA_C = {1 2 3 4 5}
Anz_Equin = {-1 1 1} JA_Equin = {1 2 3}
Anz_A = {1 -1 2 -1 1 3 3 1 -1 1 -3 2} JA_A = {1 2 4 1 2 4 1 2 4 1 2 4}
IA_A = {1 4 7 7 10 13}

Για τον εντοπισμό των κενών στηλών ελέγχουμε τον IA_A και παρατηρούμε αν επαναλαμβάνεται κάποιο νούμερο, σε περίπτωση που αυτό συμβαίνει έχουμε κενή στήλη. Εδώ η κενή στήλη είναι η 3η. Επειδή, c[3] = 1 ≥ 0 ο περιορισμός μπορεί να διαγραφεί. Αντίστοιχα για να εντοπίσουμε τις κενές γραμμές

ελέγχουμε στον πίνακα JA_A αν υπάρχει κάποιος αριθμός γραμμής ο οποίος δεν εμφανίζεται. Εδώ, κενή είναι η γραμμή είναι η 3η. Επειδή, Equin[3] = 1 και b[3] = -1 η γραμμή είναι πλεονασματική και μπορεί να διαγραφεί. Οι πίνακες μετά τις διαγραφές είναι:

Anz_B = {5 7 5} JA_B = {1 2 3}
Anz_C = {-1 2 -4 -3} JA_C = {1 2 3 4}
Anz_Equin = {-1 1} JA_Equin = {1 2}
Anz_A = {1 -1 2 -1 1 3 3 1 -1 1 -3 2} JA_A = {1 2 3 1 2 3 1 2 3 1 2 3}
IA_A = {1 4 7 10 13}

Η διαδικασία εντοπισμού κενών γραμμών και στηλών είναι γρήγορη και δεν αντιμετωπίζει πρόβλημα με τον τρόπο αποθήκευσης. Εναλλακτικά, θα μπορούσε να ενσωματωθεί στη συνάρτηση διαβάσματος του αρχείου mps όπου θα εμποδιζόταν η εγγραφή, στο ενδιάμεσο αρχείων, κόμβων της λίστας Var με κενή την λίστα ConstraintVarValue.

2.2.2 Εντοπισμός και διαγραφή γραμμών με ένα μη μηδενικό στοιχείο

Οι περιορισμοί στους οποίους εμφανίζεται μόνο μια μεταβλητή ονομάζονται singleton και σε κάποιες περιπτώσεις είναι πλεονασματικοί. Ο αρχικός διαχωρισμός των singleton γραμμών γίνεται σύμφωνα με τον τύπο του περιορισμού, στη συνέχεια αυτές με ισότητα (equin=0) θα αναφέρονται ως singleton ισοτικές γραμμές ενώ αυτές με ανισότητα (equin=-1 ή equin=1) θα ονομάζονται ανισοτικές γραμμές με ένα μη μηδενικό στοιχείο.

Όπως είναι προφανές οι singleton ισοτικές γραμμές είναι της μορφής $a_{ik}x_k=b_i$, με αποτέλεσμα να μπορεί να υπολογιστεί η τιμή της μεταβλητής $x_k=b_i/a_{ik}$. Στη συνέχεια του προβλήματος η μεταβλητή αυτή έχει πλέον σταθερή τιμή και μπορεί να διαγραφεί, αφού πρώτα η τιμή της αντικατασταθεί σε όλους τους περιορισμούς που λαμβάνει μέρος. Όμως κάθε μεταβλητή του προβλήματος πρέπει να ικανοποιεί τον περιορισμό μη αρνητικότητας $x\geq 0$, έτσι σε περίπτωση που η τιμή της x_k έχει υπολογιστεί αρνητική τότε το πρόβλημα είναι αδύνατο.

Όλα τα παραπάνω συνοψίζονται στα παρακάτω βήματα:

- Εντοπισμός singleton ισοτικής γραμμής
- Υπολογισμός $x_k=b_i/a_{ik}$
- Αν $x_k \geq 0$ τότε ο περιορισμός αυτός είναι πλεονασματικός και διαγράφεται αφού πρώτα γίνει αντικατάσταση της τιμής αυτής σε όλους του περιορισμούς που εμφανίζεται και ανανέωση του δεξιού μέρους. Σε αντίθετη περίπτωση το πρόβλημα είναι αδύνατο. (Fourer Robert και David M. Gay [9])

Παράδειγμα

Έστω το γραμμικό πρόβλημα:

$$\begin{aligned}
\min\ & -2x_1 + 3x_2 - 3x_3 + x_4 \\
\text{μ.π.}\ & x_1 - x_2 + 3x_3 \le 8 \\
& 2x_1 + 1x_3 = 10 \\
& -x_1 + 2x_2 - 3x_4 \ge -5 \\
& 2x_3 = 4 \\
& x_j \ge 0\ ,\ j = 1, 2, 3, 4
\end{aligned}$$

Στο πρόβλημα αυτό υπάρχει ένας ισοτικός singleton περιορισμός, αυτός της γραμμής 4. Σύμφωνα λοιπόν με τα παραπάνω $x_3 = 4 / 2 = 2 \ge 0$. Οπότε ο περιορισμός αυτός είναι πλεονασματικός. Κάνοντας την αντικατάσταση το γραμμικό πρόβλημα έρχεται στην παρακάτω μορφή:

$$\begin{aligned}
\min\ & -2x_1 + 3x_2 - 3*2 + x_4 \\
\text{μ.π.}\ & x_1 - x_2 + 3*2 \le 8 \\
& 2x_1 + 1*2 = 10 \\
& -x_1 + 2x_2 - 3x_4 \ge -5 \\
& 2*2 = 4 \\
& x_j \ge 0\ ,\ j = 1, 2, 4
\end{aligned}$$

Κάνοντας τις πράξεις προκύπτει,

$$\begin{aligned}
\min\ & -2x_1 + 3x_2 + x_4 - 6 \\
\text{μ.π.}\ & x_1 - x_2 \le 2 \\
& 2x1 = 8 \\
& -x_1 + 2x_2 - 3x_4 \ge -5 \\
& x_j \ge 0\ ,\ j = 1, 2, 4
\end{aligned}$$

Όμως εμφανίζεται και τώρα ένας singleton ισοτικός περιορισμός στην γραμμή 2. Μετά την απαλοιφή του με την ίδια διαδικασία προκύπτει το σύστημα:

$$\begin{aligned}
\min\ & 3x_2 + x_4 - 14 \\
\text{μ.π.}\ & -x_2 \le -2 \\
& 2x_2 - 3x_4 \ge -1 \\
& x_j \ge 0\ ,\ j = 2, 4
\end{aligned}$$

Στην παραπάνω μορφή του το γραμμικό πρόβλημα εμφανίζει στην αντικειμενική του συνάρτηση σταθερό όρο. Ο όρος αυτός διατηρείται σε μια μεταβλητή στη μνήμη του υπολογιστή και μετά την επίλυση του από τον λύτη, τον αφαιρεί από την υπολογισμένη τιμή. Σε ορισμένα

προβλήματα όπως το e226 υπάρχει αποθηκευμένος ένας σταθερός όρος στο τμήμα RHS με όνομα περιορισμού το όνομα της αντικειμενικής συνάρτησης. Σε αυτές τις περιπτώσεις η βέλτιστη λύση παίρνει την τιμή της από τον τύπο:

z-constantTerm+constantTerm0

με z την τιμή που επιστρέφει ο αλγόριθμος, constantTerm τον σταθερό όρο των προλυτικών διαδικασιών και constantTerm0 τον αρχικό σταθερό όρο.

Στις ανισοτικές γραμμές με ένα μη μηδενικό στοιχείο οι έλεγχοι διαχωρίζονται ανάλογα με τον τύπο της ανισότητας όπως φαίνεται παρακάτω:

- Αν $a_{ik}x_k \leq b_i$ τότε,
  i. Αν $a_{ik}>0$ και $b_i<0$, τότε το πρόβλημα είναι αδύνατο
  ii. Αν $a_{ik}<0$ και $b_i>0$, τότε ο περιορισμός i είναι πλεονασματικός
  iii. Αν $a_{ik}>0$ και $b_i=0$, τότε ο περιορισμός i και η μεταβλητή k είναι πλεονασματικοί
  iv. Αν $a_{ik}<0$ και $b_i=0$, τότε ο περιορισμός i είναι πλεονασματικός

- Αν $a_{ik}x_k \geq b_i$ τότε,
  i. Αν $a_{ik}>0$ και $b_i<0$, τότε ο περιορισμός i είναι πλεονασματικός
  ii. Αν $a_{ik}<0$ και $b_i>0$, τότε το πρόβλημα είναι αδύνατο
  iii. Αν $a_{ik}>0$ και $b_i=0$, τότε ο περιορισμός i είναι πλεονασματικός
  iv. Αν $a_{ik}<0$ και $b_i=0$, τότε ο περιορισμός i και η μεταβλητή k είναι πλεονασματικοί (Fourer Robert και David M. Gay [9])

Παράδειγμα

Έστω το γραμμικό πρόβλημα:

$$\min -3x_1 + 2x_2 - x_3 - 5x_4$$
$$\text{μ.π.} \quad x_1 + x_2 + 3x_3 + x_4 \leq 8$$
$$-2x_1 \geq 0$$
$$-x_1 - 3x_2 + x_3 - 2x_4 \geq 12$$
$$x_j \geq 0 \ , j = 1, 2, 3, 4$$

Όπως γίνεται εύκολα κατανοητό στο παραπάνω πρόβλημα υπάρχει ένας ανισοτικός περιορισμός με ένα μη μηδενικό στοιχείο (γραμμή 2). Σύμφωνα λοιπόν με τα παραπάνω και επειδή η ανισότητα είναι μεγαλύτερο ή ίσο, το a=-2>0 και το b=0 οπότε η γραμή2 και η στήλη 1 είναι πλεονασματικές. Μετά από την διαγραφή τους το γραμμικό πρόβλημα παίρνει την μορφή:

$$\min 2x_2 - x_3 - 5x_4$$
$$\text{μ.π.} \quad x_2 + 3x_3 + x_4 \leq 8$$
$$-3x_2 + x_3 - 2x_4 \geq 12$$
$$x_j \geq 0 \ , j = 2, 3, 4$$

Η διαδικασία αυτή είναι εξαιρετικά χρονοβόρα επειδή είναι χρονικά δαπανηρή η διαγραφή γραμμών από τις λίστες που απαρτίζουν την μήτρα Α, λόγω της αραιής αποθήκευσης.

2.2.3 Εντοπισμός και διαγραφή στηλών με ένα μη μηδενικό στοιχείο

Οι μεταβλητές οι οποίες συμμετέχουν μόνο σε έναν περιορισμό ονομάζονται singleton. Οι μεταβλητές αυτής της μορφής μπορεί να προσδιορίσουν ότι είτε το πρόβλημα είναι αδύνατο είτε ότι κάποιες γραμμές και κάποιες στήλες είναι πλεονασματικός. Ο διαχωρισμός αρχικά γίνεται σύμφωνα με τον τύπο της ανισότητας στον οποίο συμμετέχει η singleton μεταβλητή. Οι singleton μεταβλητές που συμμετέχουν σε ανισοτικό περιορισμό ονομάζονται δυϊκές ανισοτικές γραμμές με ένα μη μηδενικό στοιχείο, ενώ αυτές που συμμετέχουν σε ισοτικό περιορισμό ονομάζονται ελεύθερες στήλες με ένα μη μηδενικό στοιχείο.

Οι δυϊκές ανισοτικές γραμμές συμπεριφέρονται με τον ίδιο τρόπο με αυτές του αρχικού προβλήματος που αναφέρθηκε στην παράγραφο 2.2.2. Πιο συγκεκριμένα:

- Αν ο περιορισμός είναι της μορφής μικρότερο ή ίσο τότε,
  i. Αν $a_{ik}<0$ και $c_k<0$, τότε το πρόβλημα είναι αδύνατο
  ii. Αν $a_{ik}>0$ και $c_k>0$, τότε η μεταβλητή k είναι πλεονασματική
  iii. Αν $a_{ik}<0$ και $c_k=0$, τότε ο περιορισμός i και η μεταβλητή k είναι πλεονασματικοί
  iv. Αν $a_{ik}>0$ και $c_k=0$, τότε η μεταβλητή k είναι πλεονασματική

- Αν ο περιορισμός είναι της μορφής μεγαλύτερο ή ίσο τότε,
  i. Αν $a_{ik}<0$ και $c_k>0$, τότε η μεταβλητή k είναι πλεονασματική
  ii. Αν $a_{ik}>0$ και $c_k<0$, τότε το πρόβλημα είναι αδύνατο
  iii. Αν $a_{ik}<0$ και $c_k=0$, τότε η μεταβλητή k είναι πλεονασματική
  iv. Αν $a_{ik}>0$ και $c_k=0$, τότε ο περιορισμός i και η μεταβλητή k είναι πλεονασματικοί (Fourer Robert και David M. Gay [9])

Παράδειγμα
Έστω το γραμμικό πρόβλημα:

$$\begin{aligned} \min\ & 3x_1 + 2x_2 - x_3 - 5x_4 \\ \text{μ.π.}\ & x_2 + 3x_3 - x_4 \le 0 \\ & -2x_1 + x_2 - x_3 + 3x_4 \ge 0 \\ & -3x_2 + x_3 - 2x_4 \le 12 \\ & x_j \ge 0,\ j = 1, 2, 3, 4 \end{aligned}$$

Όπως γίνεται εύκολα κατανοητό, στο πρόβλημα υπάρχει μια στήλη με ένα μη μηδενικό στοιχείο η οποία στο δυϊκό πρόβλημα είναι ανισοτική γραμμή.

Λαμβάνοντας υπ' όψιν $a_{21}<0$ και $c_1>0$ η στήλη 1 είναι πλεονασματική. Το πρόβλημα μετά την διαγραφή της στήλης παίρνει την εξής μορφή:

$$\begin{aligned} \min\ & 2x_2 - x_3 - 5x_4 \\ \text{μ.π.}\ & x_2 + 3x_3 - x_4 \le 0 \\ & x_2 - x_3 + 3x_4 \ge 0 \\ & -3x_2 + x_3 - 2x_4 \le 12 \\ & x_j \ge 0,\ j = 2, 3, 4 \end{aligned}$$

Στην περίπτωση των ελεύθερων στηλών το πρόβλημα δεν μπορεί να αποδειχθεί αδύνατο. Αντίθετα, υπό προϋποθέσεις μπορεί να αποδειχθεί ότι η μεταβλητή και ο περιορισμός στον οποίο εμφανίζεται είναι πλεονασματικοί και μπορούν να διαγραφούν. Έστω ο περιορισμός,

$a_{i1}x_1 + \ldots + a_{is}x_2 + \ldots + a_{in}x_n = b$ με s την singleton στήλη.

Η στήλη s και η γραμμή i είναι πλεονασματικές αν:

- $a_{is} > 0$ και $a_{ij} \le 0$ για $j \ne s$ ή
- $a_{is} < 0$ και $a_{ij} \ge 0$ για $j \ne s$

Στην περίπτωση που ισχύει κάποια από τις δύο παραπάνω σχέσεις, η γραμμή και η στήλη μπορούν να διαγραφούν. Στη περίπτωση πού $c_s = 0$ δεν υπάρχει καμία άλλη τροποποίηση στο πρόβλήμα. Σε αντίθετη περίπτωση όμως πρέπει να γίνει μια περιστροφή στο στοιχείο $a_{is} \ne 0$. Η περιστροφή αυτή έχει ως αποτέλεσμα να γίνει το στοιχείο $c_s = 0$ (Fourer Robert και David M. Gay [9]). Το διάνυσμα της αντικειμενικής συνάρτησης ανανεώνεται σύμφωνα με τον τύπο:

$c[j] = c[j] - (c_s / a_{is}) * a[i][j]$ και ο σταθερός όρος
$constantTerm = constantTerm - (c_s / a_{is}) * b[i]$

```
// Update c
                long double aaa;
                c=JA_C->find(tmp1);
                c=Anz_C->getNth(c);
                if (c!=0) {
                  aaa=c/a;
                  constantTerm_=constantTerm_-(aaa*b);
                  LinkedList<long double> *TempC_Anz=new
LinkedList<long double>;
                  LinkedList<long double> *TempC_JA=new
LinkedList<long double>;
                  Temp=new LinkedList<long double>;
                  TempList=new LinkedList<long double>;
                  Temp1=new LinkedList<long double>;
                  int start=1;
                  int rows1,apo;
                  int col=0;
                  int finnish1=0;
                  IA_A->removeHead();
                  Temp1->addTail(1);
                  count=0;
```

```
                while (1) {
                  count++;
                  rows1=IA_A->getNth(1);
                  finnish1=rows1-col;
                  if (finnish1<0) break;
                  IA_A->removeHead();
                  Temp1->addTail(rows1);
                  for (int j=start;j<finnish1;j++) {
                     temp1=JA_A->getNth(1);
                     value=Anz_A->getNth(1);
                     if (value==0) break;
                     if (temp1==tmp2){
                        TempC_Anz->addTail(value*aaa);
                        TempC_JA->addTail(count);
                     }
                     Temp->addTail(value);
                     TempList->addTail(temp1);
                     Anz_A->removeHead();
                     JA_A->removeHead();
                  }
                  col=col+finnish1-1;
                }
                IA_A->setList(Temp1);
                Anz_A->setList(Temp);
                JA_A->setList(TempList);
                while(1){
                  temp1=TempC_JA->getNth(1);
                  value=TempC_Anz->getNth(1);
                  if (temp1==0) break;
                  aaa=JA_C->find(temp1);
                  if (aaa==0) {
                     value=-value;
                     Anz_C->addTail(value);
                     JA_C->addTail(temp1);
                  } else {
                     value=Anz_C->getNth(aaa)-value;
                     Anz_C->setNth1(aaa,value);
                  }
                  TempC_JA->removeHead();
                  TempC_Anz->removeHead();
                }
              }
```

Όπως γίνεται αντιληπτό από τον παραπάνω κώδικα η διαδικασία είναι χρονοβόρα προγραμματιστικά καθώς και η εκτέλεση του προγράμματος είναι αρκετά αργή.

Παράδειγμα

Έστω το γραμμικό πρόβλημα:

$$\min -2x_1 - x_2 - 3x_3 - x_4$$

$$\text{μ.π.} \quad x_1 + 3x_2 + x_4 \leq 10$$

$$2x_1 + x_2 - 2x_3 + 3x_4 = -5$$

$$-x_1 + 2x_2 + x_4 \leq 9$$

$$x_j \geq 0 \ , j = 1, 2, 3, 4$$

Στο παραπάνω παράδειγμα ελεύθερη μεταβλητή είναι η 3η, και επειδή η στήλη 3 είναι ετερόσημη με όλες τις άλλες στην γραμμή 2 η γραμμή 2 και η στήλη 3 είναι πλεονασματικές. Επειδή, το $c_2 \neq 0$ πρέπει να εκτελεστεί μια περιστροφή. Έτσι, $c = c – (c_3/a_{23})*a_{i.}$ = [-2 -1 -3 -1] – (-3/-2) [2 1 -2 3] =
[-2 -1 -3 -1] – [3 3/2 -3 9/2] =
[-5 -5/2 0 -11/2]
constant Term = constant Term – $(c_3/a_{23})*b_2$ =
0 – (-3/-2)(-5) = 15/2. Το γραμμικό πρόβλημα παίρνει την παρακάτω μορφή:*

$$\min -5x_1 - (5/2)x_2 - (11/2)x_4$$
$$\text{μ.π.} \quad x_1 + 3x_2 + x_4 \leq 10$$
$$-x_1 + 2x_2 + x_4 \leq 9$$
$$x_j \geq 0 \ , j = 1, 2, 4$$

2.2.4 Εντοπισμός και διαγραφή πλεονασματικών ορίων

Σε ορισμένα προβλήματα, οι περιορισμοί είναι πλεονασματικοί σε περίπτωση που τα όρια του περιορισμού υπερκαλύπτονται από τα όρια της μεταβλητής. Το γεγονός αυτό συμβαίνει στις παρακάτω περιπτώσεις:

- Αν ένας περιορισμός είναι της μορφής μικρότερο ή ίσο, όλοι οι συντελεστές μεταβλητών του περιορισμού είναι μικρότεροι ή ίσοι του μηδενός και το b είναι μεγαλύτερο ή ίσο με το μηδέν.
- Αν ο περιορισμός είναι της μορφής μεγαλύτερο ή ίσο, όλοι οι συντελεστές μεταβλητών του περιορισμού είναι μεγαλύτεροι ή ίσοι του μηδενός και το b είναι μικρότερο ή ίσο του μηδενός.

Παράδειγμα
Έστω το γραμμικό πρόβλημα:

$$\min -3x_1 - 5x_2 - 10x_3 - 2x_4$$
$$\text{μ.π.} \quad x_1 + 3x_2 + 2x_3 + x_4 \leq 10$$
$$-x_1 - 3x_2 - x_3 - 5x_4 \leq 6$$
$$x_3 + 3x_4 \geq -5$$
$$x_1 + 5x_2 + 2x_3 + x_4 \geq -8$$
$$x_j \geq 0 \ , j = 1, 2, 3, 4$$

Στο παραπάνω παράδειγμα υπάρχουν ανισοτικοί περιορισμοί, στη συνεχεία πρέπει να ελεγχθούν όλοι οι περιορισμοί για πλεονασματικά όρια σύμφωνα με τους δύο κανόνες που αναφέρθηκαν παραπάνω. Οι περιορισμοί μικρότερο ή ίσο πρέπει να έχουν $b \geq 0$, συνθήκη που ικανοποιούν και οι δύο, αλλά ο περιορισμός 1 δεν ικανοποιεί την συνθήκη που επιβάλει όλοι οι συντελεστές μεταβλητών να είναι

μικρότεροι του μηδενός. Οπότε μπορεί να διαγραφεί μόνο ο περιορισμός 2. Μετά το τέλος των ελέγχων με equin=-1, το γραμμικό πρόβλημα έχει την παρακάτω μορφή:

$$\min -3x_1 - 5x_2 - 10x_3 - 2x_4$$
$$\text{μ.π.} \quad x_1 + 3x_2 + 2x_3 + x_4 \leq 10$$
$$x_3 + 3x_4 \geq -5$$
$$x_1 + 5x_2 + 2x_3 + x_4 \geq -8$$
$$x_j \geq 0 \ , j = 1, 2, 3, 4$$

Στη συνέχεια ελέγχονται οι περιορισμοί με equin=1. Οι περιορισμοί αυτοί πρέπει να έχουν $b \leq 0$ και οι συντελεστές των μεταβλητών να είναι θετικοί αριθμοί. Παρατηρούμε λοιπόν ότι και οι δύο περιορισμοί ικανοποιούν τους περιορισμούς, με αποτέλεσμα να μπορούν να διαγραφούν.

$$\min -3x_1 - 5x_2 - 10x_3 - 2x_4$$
$$\text{μ.π.} \quad x_1 + 3x_2 + 2x_3 + x_4 \leq 10$$
$$x_j \geq 0 \ , j = 1, 2, 3, 4$$

Η συγκεκριμένη προλυτική διαδικασία είναι η πιο δαπανηρή χρονικά, αφού για κάθε ανισοτικό περιορισμό απαιτεί να προσπελαστούν όλα τα στοιχεία της γραμμής ενώ αποθήκευση είναι προσανατολισμένη σε προσπέλαση κατά στήλες.

2.2.5 Εντοπισμός και διαγραφή πλεονασματικών μεταβλητών

Οι μεταβλητές x_j του ισοτικού περιορισμού

$$a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n = 0$$

με i =1 , 2 , ... , m ονομάζεται πλεονασματικές αν ισχύει είτε $a_{ij} \geq 0$ είτε $a_{ij} \leq 0$. Και στις δύο περιπτώσεις οι μεταβλητές x_j είναι πλεονασματικές και μπορούν να διαγραφούν. Η συγκεκριμένη προλυτική διαδικασία στην εφαρμογή εκτελείται στο αρχικό πρόβλημα, ενώ περισσότερες τέτοιες μεταβλητές εμφανίζονται μετά την διαδικασία προσθήκης χαλαρών μεταβλητών.

Σύμφωνα με τον Παπαρίζο [7] μια αποτελεσματική εφαρμογή του συγκεκριμένου αλγορίθμου είναι η εκτέλεση της συγκεκριμένης προλυτικής διαδικασίας στη συνάρτηση lprref κατά την οποία εκτελούνται γραμμοπράξεις για τον σχηματισμό της μοναδιαίας μήτρας στη επαυξημένη μήτρα [A|b] μετά την προσθήκη χαλαρών μεταβλητών (Tomlin, Welch [18]).

Παράδειγμα

Έστω το γραμμικό πρόβλημα:

$$\begin{aligned}
&\min -3x_1 + 2x_2 - x_3 - 4x_4 \\
&\text{μ.π.} \quad x_1 - 2x_2 + x_3 + x_4 = 10 \\
&\qquad -2x_1 + x_2 - 3x_3 - x_4 \le -7 \\
&\qquad -2x_1 + 5x_2 + 2x_3 - 2x_4 \le -20 \\
&\qquad 5x_1 - 2x_2 - x_3 + x_4 = 8 \\
&\qquad x_j \ge 0 \ , j = 1, 2, 3, 4
\end{aligned}$$

το πρόβλημα μετά την προσθήκη χαλαρών μεταβλητών τροποποιείται:

$$\begin{aligned}
&\min -3x_1 + 2x_2 - x_3 - 4x_4 \\
&\text{μ.π.} \quad x_1 - 2x_2 + x_3 + x_4 = 10 \\
&\qquad -2x_1 + x_2 - 3x_3 - x_4 + x5 = -7 \\
&\qquad -2x_1 + 5x_2 + 2x_3 - 2x_4 + x6 = -20 \\
&\qquad 5x_1 - 2x_2 - x_3 + x_4 = 8 \\
&\qquad x_j \ge 0 \ , j = 1, 2, \ldots, 6
\end{aligned}$$

Ο επαυξημένος πίνακας [A | B] στον οποίο θα εφαρμοστούν γραμμοπράξεις είναι ο εξής:

1	*-2*	*1*	*1*	*0*	*0*	*10*	
-2	*1*	*-3*	*-1*	*1*	*0*	*-7*	
-2	*5*	*2*	*-2*	*0*	*1*	*-20*	→
5	*-2*	*-1*	*1*	*0*	*0*	*8*	

1	*-2*	*1*	*1*	*0*	*0*	*10*	
0	*-3*	*-1*	*1*	*1*	*0*	*13*	
0	*1*	*4*	*0*	*0*	*1*	*0*	→
0	*8*	*-6*	*-4*	*0*	*0*	*-42*	

Σε αυτή τη φάση ο τρίτος περιορισμός είναι της επιθυμητής μορφής οπότε οι μεταβλητές x_2, x_3 και x_6 μπορούν να διαγραφούν.

1	*-2*	*1*	*1*	*0*	*0*	*10*	
0	*-3*	*-1*	*1*	*1*	*0*	*13*	→
0	*1*	*4*	*0*	*0*	*1*	*0*	
0	*-2*	*3/2*	*1*	*0*	*0*	*13*	

1	*0*	*-1/2*	*0*	*0*	*0*	*-3*
0	*-1*	*-5/2*	*0*	*1*	*0*	*0*
0	*1*	*4*	*0*	*0*	*1*	*0*
0	*-2*	*3/2*	*1*	*0*	*0*	*13*

Η διαδικασία των γραμμοπράξεων έχει ολοκληρωθεί. Η γραμμή 2 παρ' ότι έχει σαν δεξιό μέρος το 0, δεν έχει ομόσημους όλους τους συντελεστές των μεταβλητών. Οπότε διαγράφονται μόνο οι μεταβλητές x_2, x_3 και x_6. Το πρόβλημα παίρνει την παρακάτω μορφή:

$$\begin{aligned} \min\ & -3x_1 - 4x_4 \\ \text{μ.π.}\ & x_1 + x_4 = 10 \\ & -2x_1 - x_4 \leq -7 \\ & -2x_1 - 2x_4 \leq -20 \\ & 5x_1 + x_4 = 8 \\ & x_j \geq 0\ ,\ j = 1, 4 \end{aligned}$$

Όπως αναφέρθηκε παραπάνω, η διαδικασία αυτή εφαρμόστηκε μόνο στα αρχικά δεδομένα με αποτέλεσμα να μην έχει ιδιαίτερη επιτυχία. Για την ακρίβεια διαγράφηκαν μεταβλητές μόνο σε δύο από τα 66 προγράμματα, ποσοστό 3.03%. Ο εντοπισμός των μεταβλητών δεν είναι ιδιαίτερα δαπανηρή διαδικασία όσο η διαγραφή των μεταβλητών, λαμβάνοντας υπ' όψιν ότι το πρόβλημα woodw εντόπισε και διέγραψε 1544 μεταβλητές.

2.3 Ανοχές

Μετά την ολοκλήρωση των προλυτικών διαδικασιών ο προγραμματιστής πρέπει να ορίσει κάποια τιμή η οποία στη συνέχεια του προγράμματος θα θεωρείται η μικρότερη δυνατή που μπορεί να εμφανιστεί. Η τιμή αυτή ονομάζεται ανοχή (tolerance). Έτσι, οποιαδήποτε τιμή είναι κατά απόλυτη τιμή μικρότερη από την ανοχή θα τίθεται ίση με μηδέν. Η χρήση ανοχών είναι απαραίτητη για τη λύση των περισσότερων προβλημάτων, με αυτόν τον τρόπο αποφεύγονται τα σφάλματα στρογγυλοποίησης που προέρχονται από την περιορισμένη ακρίβεια των υπολογιστών.

Η χρήση ανοχών απαιτεί μεγάλη προσοχή, γιατί λάθος επιλογή στην τιμή τους μπορεί να προκαλέσει στην εκτέλεση περισσότερων επαναλήψεων, στην εύρεση λάθος τιμής αντικειμενικής συνάρτησης, ή ακόμα και στην μη επίλυση του προβλήματος (Σαμαράς [8]). Γι' αυτό το λόγο υπολογίζεται από τα δεδομένα του προβλήματος μια προτεινόμενη τιμή ανοχής από τον τύπο tol=max (rows,cols)*eps*norm(A). Όμως ακόμα και η τιμή αυτή δεν είναι βέβαιο ότι φέρνει τα καλύτερα αποτελέσματα και η καλύτερη τιμή βρίσκεται με δοκιμές.

```
tol=DBL_EPSILON*norm(a,rows,cols);
for (k=0;k<rows;k++) {
   if (a[k][j]!=0) {
       long double pivot1=a[k][j];
       if (k!=i) {
           value=rhs[k]-rhs[i]*pivot1;
           if (fabs(value)>tol) rhs[k]=value;
                   else rhs[k]=0;
       }
```

ΚΕΦΑΛΑΙΟ 3
ΑΛΓΟΡΙΘΜΟΣ SIMPLEX

Ο πρωτεύων αλγόριθμος Simplex είναι ο πρώτος που κατασκευάστηκε για την επίλυση του γραμμικού προβλήματος. Στο κεφάλαιο αυτό εξετάζεται το θεωρητικό υπόβαθρο για την ανάπτυξη γενικών κριτηρίων βελτιστότητας, τα ειδικής μορφής σημεία που κατασκευάζονται από τους αλγορίθμους simplex και η γενική μεθοδολογία των αλγορίθμων. Στο τέλος του κεφαλαίου παρουσιάζονται ειδικοί τρόποι αντιστροφής ενός πίνακα με σκοπό την βελτίωση του χρόνου εκτέλεσης του προγράμματος.

3.1 Αλγόριθμος Simplex

Θεωρώντας το γενικό γραμμικό πρόβλημα στη μορφή Γ.Π.1, το αντίστοιχο δυϊκό του πρόβλημα ορίζεται ως εξής:

$$\begin{aligned} &max\ w^T b \\ &\mu.\pi.\ A^T w + s = 0 \\ &s \geq 0 \end{aligned} \qquad \text{(Γ.Π. 2)}$$

Διαμερίζοντας την μήτρα A ως (B N) και τα διανύσματα x και c το αρχικό πρόβλημα γράφετε ως εξής:

$$\begin{aligned} &max\ z = c_B^T x_B + c_N^T x_N \\ &\mu.\pi.\ Bx_B + Nx_N = b \\ &x_B, x_N \geq 0 \end{aligned} \qquad \text{(Γ.Π.3)}$$

Το σύνολο δεικτών B ονομάζεται βάση αν η αντίστοιχη μήτρα B είναι αντιστρέψιμη. Η μήτρα B, η οποία είναι μεγέθους mxm ονομάζεται βασική μήτρα. Οι μεταβλητές που συμμετέχουν στον πίνακα B ονομάζονται βασικές μεταβλητές. Οι υπόλοιπες οι οποίες ανήκουν στο σύνολο δεικτών N ονομάζονται μη βασικές. Η διαμέριση αυτή ονομάζεται βασική και η λύση που αντιστοιχεί σε αυτή υπολογίζεται από τον τύπο:

$$x = [x_B\ x_N] = [B^{-1}b\ 0]$$

Η παραπάνω λύση ονομάζεται βασική λύση. Η λύση αυτή είναι εφικτή αν ικανοποιεί όλους τους περιορισμούς του γραμμικού προβλήματος. Δηλαδή, ένα βασικό σημείο είναι εφικτό αν ισχύει $x_B \geq 0$.

Η λύση του δυϊκού προβλήματος που αντιστοιχεί στη βάση B, δίνεται από την σχέση $s= c - A^T w$ με w να είναι οι πολλαπλασιαστές

simplex και υπολογίζονται από τη σχέση $w^T = (c_B)^T B^{-1}$. Η βάση B είναι δυϊκά εφικτή αν ισχυεί $s \geq 0$. Μεταξύ των λύσεων x του πρωτεύοντος προβλήματος και s του δυϊκού ισχύει ότι μια βασική λύση είναι βέλτιστη αν ισχύει $x_B \geq 0$ και $s_N \geq 0$.

Κάθε αλγόριθμος τύπου simplex κατασκευάζει μια σειρά από βασικές λύσεις B. Αν για τη βάση αυτή ισχύει $x_B \geq 0$ και $s_N \geq 0$ τότε αυτή είναι βέλτιστη και όλοι οι υπολογισμοί σταματούν. Διαφορετικά επιλέγονται δύο στήλες μια από την μήτρα B και μία από τη μήτρα N. Σε περίπτωση που δεν βρεθεί μεταβλητή να εξαχθεί από την B τότε το πρόβλημα είναι απεριόριστο και οι υπολογισμοί σταματούν. Έστω k ο δείκτης της εξερχόμενης μεταβλητής και l ο δείκτης της εισερχόμενης. Σε περίπτωση που η ανταλλαγή μπορεί να γίνει χρησιμοποιούνται οι παρακάτω σχέσεις:

$$Bnew \leftarrow Bold \setminus \{k\} \cup \{l\}$$
$$Nnew \leftarrow Nold \setminus \{l\} \cup \{k\},$$

με Bnew, Bold, Nnew και Nold τα διανύσματα δεικτών

Οι παραπάνω σχέσεις δηλώνουν ότι η βασική μεταβλητή x_k θα γίνει μη βασική και η μη βασική x_l θα γίνει βασική. Η διαδικασία αυτή της εναλλαγής ονομάζεται περιστροφή (pivoting). Η μεταβλητή που αφήνει την βάση ονομάζεται εξερχόμενη (leaving variable) και η μεταβλητή που μπαίνει στη βάση εισερχόμενη (entering variable). Σε κάθε επανάληψη οι αλγόριθμοι τύπου simplex εναλλάσσουν μια στήλη του B με μία στήλη του N. Για τον λόγο αυτό υπάρχει η περίπτωση να εμφανίζεται το φαινόμενο της κύκλωσης (cycling)*. Για την αποφυγή του φαινομένου αυτού έχουν αναπτυχθεί αρκετοί κανόνες αντικύκλωσης (Bland [10]).

Όλοι οι αλγόριθμοι simplex απαρτίζονται από τέσσερα βήματα:

1. Αρχικοποίηση (initialization). Στο βήμα αυτό υπολογίζονται όλες οι μεταβλητές οι οποίες είναι απαραίτητες για να αρχίσει να εκτελείται ο αλγόριθμος
2. Έλεγχος βελτιστότητας (optimality test). Πραγματοποίηση ελέγχου βελτιστότητας, Αν το τρέχον βασικό σημείο είναι βέλτιστο οι υπολογισμοί σταματούν και τερματίζεται ο αλγόριθμος
3. Επιλογή εισερχόμενης και εξερχόμενης μεταβλητής (choice of entering and leaving variable). Η επιλογή της εισερχόμενης και εξερχόμενης μεταβλητής μπορεί να γίνει εφαρμόζοντας διαφορετικούς κανόνες περιστροφής. Η ανταλλαγή αυτή των μεταβλητών γεωμετρικά σημαίνει μετακίνηση από ένα βασικό σημείο σε ένα άλλο.

* Κύκλωση είναι το φαινόμενο κατά το οποίο είναι δυνατόν να κατασκευαστούν δύο ίδιες βάσεις

4. Περιστροφή (pivoting). Στο βήμα αυτό υπολογίζεται η νέα βάση, οι τιμές των βασικών μεταβλητών καθώς και οι δυϊκές χαλαρές μεταβλητές.

Μετά την εκτέλεση του βήματος 4, επαναλαμβάνονται τα βήματα 2 μέχρι 4 έως ότου να εντοπιστεί αν υπάρχει βέλτιστη λύση. Για τον λόγο αυτό οι αλγόριθμοι ονομάζονται επαναληπτικοί (iterative algorithms).

3.1.1 Αναθεωρημένος πρωτεύων αλγόριθμος Simplex

Ο αλγόριθμος που αναφέρθηκε παραπάνω θεωρεί ότι για το συγκεκριμένο γραμμικό πρόβλημα υπάρχει ένα γνωστό εφικτό βασικό σημείο. Παρ' ότι αυτό δεν συμβαίνει στην πραγματικότητα στη συνέχεια αυτής της παραγράφου θα συνεχίσουμε να θεωρούμε ότι το σημείο αυτό υπάρχει χάριν απλότητας.

Ο αναθεωρημένος αλγόριθμος simplex (revised simplex algorithm) κατασκευάζει σε κάθε επανάληψη το παρακάτω ταμπλό το οποίο ονομάζεται αναθεωρημένο ταμπλό simplex (revised simplex tableau).

Basis Inversion	**RHS**
$w = c_B B^{-1}$	$z = wb$
B^{-1}	$x_B = B^{-1}b$

Ο αλγόριθμος αυτός ξεκινά τους υπολογισμούς με το τμήμα αντιστροφής βάσης έχοντας μηδενικά στη γραμμή κόστους $w = 0$ και τη μοναδιαία μήτρα στις υπόλοιπες γραμμές. Σε αντίθεση με το εκτεταμένο ταμπλό, στο αναθεωρημένο αποθηκεύονται μόνο οι απαραίτητες πληροφορίες για την εκτέλεση του αλγορίθμου. Πιο συγκεκριμένα, διατηρώντας στη μνήμη τα αρχικά δεδομένα του προβλήματος στις μήτρες A, b και c μπορούμε κάθε φορά να υπολογίζουμε τη βασική μήτρα B^{-1} και τα σύνολα δεικτών B και N. Στον παρακάτω πίνακα φαίνονται οι ποσότητες που υπολογίζονται χρησιμοποιώντας τη μήτρα B^{-1}.

Ποσότητα	**Τύπος**
Τιμές βασικών μεταβλητών	$x_B = B^{-1}b$
Τιμή αντικειμενικής συνάρτησης	$z = c_B B^{-1}b$
Στήλη περιστροφής	$h_l = B^{-1}A_{.l}$
Δυϊκές χαλαρές μεταβλητές	$s_N = c_N - c_B B^{-1}N$

Σύμφωνα με τον γενικό τρόπο λειτουργίας του αλγορίθμου σε κάθε επανάληψη πρέπει να επιλεγεί μια εισερχόμενη και μια εξερχόμενη μεταβλητή. Για την επιλογή εισερχόμενης μεταβλητής υπάρχουν πολλές στρατηγικές, οι οποίες χωρίζονται σε δύο κατηγορίες. Στη πρώτη, γίνεται μια προσπάθεια να μειωθεί ο αριθμός των επαναλήψεων μέσω της

τροποποίησης του κριτηρίου επιλογής της εισερχόμενης που προτάθηκε από τον Dantzig. Το κριτήριο αυτό είναι $s_l = \min \{s_j: s_j<0 \text{ για } j \in N\}$. Η εργασία όμως αυτή απαιτεί περισσότερη υπολογιστική εργασία ανά επανάληψη. Η δεύτερη κατηγορία επιδιώκει να μειώσει τη μέση εργασία ανά επανάληψη, αλλά έχει ως μειονέκτημα την αύξηση του συνολικού αριθμού επαναλήψεων.

Στη πρώτη κατηγορία ανήκουν οι στρατηγικές της πιο κατηφορικής ακμής, κύριοι εκπρόσωποι της οποίας είναι το σχήμα DEVEX και η μέθοδος που προτάθηκε από τους Goldfarb και Reid [15]. Ο υπολογισμός των δυϊκών χαλαρών μεταβλητών αντιπροσωπεύει n-m πράξεις εσωτερικού γινομένου και είναι χρονοβόρος σε μεγάλα γραμμικά προβλήματα. Το μειονέκτημα αυτό προσπαθούν να καλύψουν οι στρατηγικές της δεύτερης κατηγορίας. Χαρακτηριστική στρατηγική αυτής της κατηγορίας αποτελεί η μερική τιμολόγηση (partial pricing). Στη μέθοδο αυτή εξετάζεται μόνο ένα μέρος των μεταβλητών στον δείκτη N για είσοδο στη βάση. Το τμήμα που εξετάζεται αλλάζει σε κάθε επανάληψη. Σύμφωνα με τον Bixby ο συνδυασμός στρατηγικών «πιο κατηφορικής ακμής» και «μερικής τιμολόγησης» μπορεί να επιφέρει αρκετά καλά αποτελέσματα.

Η επιλογή της εξερχόμενης μεταβλητής πραγματοποιείται με το τεστ ελαχίστου λόγου (minimum ratio test). Σύμφωνα με αυτό η εξερχόμενη μεταβλητή υπολογίζεται από τη σχέση:

$$x_k = x_{B[r]} = (B^{-1}b)_r / h_{rl} =$$
$$= \min \{(B^{-1}b)_i / h_{rl} : h_{rl} > 0, i = 0, 1, 2, \dots , m\}$$
$$\text{με } h = B^{-1}A_{.l}$$

Σε περίπτωση που δεν υπάρχει $h_{rl} > 0$ το γραμμικό πρόβλημα είναι απεριόριστο. Ο αριθμός r ονομάζεται δείκτης περιστροφής ενώ το στοιχείο h_{rl} ονομάζεται στοιχείο περιστροφής (pivot element).

Στην περίπτωση κατά την οποία περισσότεροι του ενός δείκτες ικανοποιούν το τεστ ελαχίστου λόγου, οι δείκτες αυτοί ονομάζονται επιλέξιμοι (eligible) και άρα υπάρχει δεσμός (tie). Σε περίπτωση που σε κάθε επανάληψη επιλέγεται τυχαία μια μεταβλητή μπορεί να εμφανιστεί το φαινόμενο της κύκλωσης, με αποτέλεσμα τον μη τερματισμό του αλγορίθμου. Στη βιβλιογραφία εμφανίζονται αρκετές τεχνικές για το σπάσιμο των δεσμών. Στη υλοποίηση του αναθεωρημένου αλγορίθμου simplex που πραγματοποιήθηκε χρησιμοποιήθηκε ο κανόνας επιλογής «σε κάθε επανάληψη επιλέγεται ο δείκτης με την μικρότερη τιμή». Δηλαδή,

$k = \min \{k_1, k_2, \dots, k_p\}$, με $p \leq m$ το αριθμών των επιλέξιμων μεταβλητών.

Όλοι οι αλγόριθμοι τύπου simplex έχουν ένα μεγάλο μειονέκτημα σε ότι αφορά τον CPU χρόνο που απαιτούν. Ο υπολογισμός της αντίστροφης της βάσης είναι εξαιρετικά χρονοβόρος και αποτελεί το 70% – 80% του συνολικού χρόνου εκτέλεσης του αλγορίθμου. Για τον λόγο αυτό και επειδή ο υπολογισμός του αντίστροφου με κάποια μέθοδο με γραμμοπράξεις (όπως Gauss-Jordan [6]) είναι αργές στη εφαρμογή χρησιμοποιήθηκαν δύο προσεγγιστικές μέθοδοι που υπολογίζουν τον αντίστροφο από τον αντίστροφο πίνακα της προηγούμενης επανάληψης. Οι μέθοδοι αυτοί περιγράφονται εκτενώς στη παράγραφο 3.3.

Η τεχνική της ανανέωσης σε σύγκριση με τον υπολογισμό κάποιων διανυσμάτων είναι αρκετά πιο γρήγορη με αποτέλεσμα εκτός από την ανανέωση της μήτρας B^{-1} να ανανεώνονται αντί να υπολογίζονται και τα διανύσματα x_B και s_N. Η ανανέωση γίνεται με βάση τους παρακάτω τύπους:

$f = x_{B[r]}$
$g = h_r$
$a = s_{N[t]}$
$h_r = -1$
$x_{B[r]} = 0$
$s_{N[t]} = 0$
$h = B^{-1}A_{.l}$
$H_{rN} = B^{-1}{}_{r.} A_N$
$x_B = x_B - (f/g)*h$
$s_N = s_N - (a/g) H_{rN}$

<u>*Βηματική Περιγραφή Αναθεωρημένου Αλγορίθμου Simplex*</u>

1. *Αρχικοποίηση. Ξεκινά με μια εφικτή βασική διαμέριση (B,N). Υπολόγισε την μήτρα B^{-1} και τα διανύσματα x_B, s_N και w^T.*
2. *Έλεγχος βελτιστότητας. Αν $s_N \geq 0$ ο αλγόριθμος σταματά το πρόβλημα είναι βέλτιστο*
3. *Επιλογή εισερχόμενης-εξερχόμενης μεταβλητής. Επιλογή εισερχόμενης μεταβλητής από τη σχέση $s_l = \min \{s_j: s_j<0$ για $j \in N\}$ και εξερχόμενης από τη σχέση $x_k = x_{B[r]} = (B^{-1}b)_r/h_{rl} = min \{(B^{-1}b)_i/h_{rl} : h_{rl} > 0, i = 0, 1, 2, \dots , m\}$ με $h = B^{-1}A_{.l}$. Αν $h_{rl} \leq 0$ το πρόβλημα είναι απεριόριστο*
4. *Περιστροφή. Ανανέωση των συνόλων δεικτών B και N. Υπολογισμός μήτρας B^{-1} και των διανυσμάτων x_B, s_N και w^T. Επιστροφή στο βήμα 2*

Υπολογισμός x_B

```
xb=(long double *) new long double[rowsAfter];

for (i=0;i<rowsAfter;i++) {
          value=0;
          for (j=0;j<rowsAfter;j++)
             if (binv[i][j]!=0 && rhs[j]!=0)
                 value=value+(binv[i][j]*rhs[j]);
          if (fabs(value)>tol) xb[i]=value;
            else xb[i]=0;
}
```

Υπολογισμός s_N και w^T

```
wt=(long double *) new long double[rowsAfter];
sn=(long double *) new long double[columnsAfter-rowsAfter];

 for (j=0;j<rowsAfter;j++) {
          value=0;
          for (i=0;i<rowsAfter;i++) {
             mini=b_index[i];
             min=c[mini];
             if (binv[i][j]!=0 && min!=0)
                 value=value+(binv[i][j]*min);
          }
          if (fabs(value)>tol) wt[j]=value;
             else wt[j]=0;
       }
```

```
       for (j=0;j<columnsAfter-rowsAfter;j++) {
          value=0;
          for (i=0;i<rowsAfter;i++) {
             if (n[i][j]!=0 && wt[i]!=0)
                 value=value+(n[i][j]*wt[i]);
          }
          if (fabs(value)>tol) sn[j]=value;
            else sn[j]=0;
       }
       for (i=0;i<columnsAfter-rowsAfter;i++) {
          mini=n_index[i];
          min=c[mini];
          sn[i]=min-sn[i];
       }
```

Έλεγχος βελτιστότητας - επιλογή εισερχόμενης μεταβλητής

```
        min=LONG_MAX;
        mini=-1;
        for (i=0;i<columnsAfter-rowsAfter;i++) {
           if (sn[i]<min) {
              min=sn[i];
              mini=i;
           }
        }
        if (min>=0) stop=true;
           else {
             imported_n_index=mini;
             imported=n_index[imported_n_index];
             stop=false;
           }
```

Επιλογή εξερχόμενης μεταβλητής

```
           for (i=0;i<rowsAfter;i++) d[i]=a[i][imported];
           for (i=0;i<rowsAfter;i++) {
             value=0;
             for (j=0;j<rowsAfter;j++) {
                if (binv[i][j]!=0 && d[j]!=0)
                    value=value+(binv[i][j]*d[j]);
             }
             if (fabs(value)>tol) h[i]=value;
                else h[i]=0;
           }
           min=LONG_MAX;
           mini=-1;
           bool infeasible=true;
           for (i=0;i<rowsAfter;i++) {
               if (h[i]!=0 && fabs(h[i])>tol) {
                  if (h[i]>0 && xb[i]/h[i]<=min) {
                  if (xb[i]/h[i]==min) {
                      if (b_index[i]<b_index[mini]) {
                         min=xb[i]/h[i];
                         mini=i;
                      }
               } else {
                      min=xb[i]/h[i];
                      mini=i;
                      infeasible=false;
               }
               }
               }
           }
if (infeasible==true) {e=getCPUTime()+algTime; algTime=(e-
s); return(2);}   else {
        exported_b_index=mini;
        exported=b_index[exported_b_index];
   }
```

Περιστροφή

```
b_index[exported_b_index]=imported;
n_index[imported_n_index]=exported;
for (i=0;i<rowsAfter;i++) d[i]=b[i][exported_b_index];
for (i=0;i<rowsAfter;i++)
            b[i][exported_b_index]=n[i][imported_n_index];
for (i=0;i<rowsAfter;i++) n[i][imported_n_index]=d[i];
```

Υπολογισμός βέλτιστης λύσης

```
value=0;
for (i=0;i<rowsAfter;i++) {
     mini=b_index[i];
     min=c[mini];
     value=value+(xb[i]*min);
}
if (fabs(value)>tol) z=value;
     else z=0;
```

3.1.2 Σύγκριση κλασικού και αναθεωρημένου αλγόριθμου Simplex

Είναι φανερό ότι ο κλασσικός αλγόριθμος simplex όπως παρουσιάστηκε από τον Dantzig [11] δεν ήταν η καταλληλότερη μορφή για υλοποίηση σε ηλεκτρονικό υπολογιστή. Και αυτό γιατί χρειαζόταν να αποθηκεύσει στη μνήμη μια μήτρα (m+1)x(n+1) όπου οι αρχικές διαστάσεις του πίνακα A είναι mxn. Οι επιπλέον γραμμή και στήλη αναπαριστούν την αντικειμενική συνάρτηση και το δεξιό μέρος αντίστοιχα. Για το λόγο αυτό βρέθηκε ο αναθεωρημένος αλγόριθμος simplex. Ο Wagner [19] σε μια συγκριτική μελέτη των δύο αλγορίθμων απέδειξε ότι ο αναθεωρημένος αλγόριθμος είναι καλύτερος αν n > 3m. Σε αυτόν αποθηκεύεται μία μήτρα (m+1) x (m+1). Ο αριθμός των πολλαπλασιασμών, διαιρέσεων, προσθέσεων και αφαιρέσεων που απαιτούνται στις δύο μορφές φαίνονται στον παρακάτω πίνακα:

Πράξεις	**Κλασική**	**Αναθεωρημένη**
Πολλαπλασιασμοί	m(n-m) + n +1	m(n+2) +1
Διαιρέσεις		
Προσθέσεις – Αφαιρέσεις	m(n-m+1)	m(n+1)
Σύνολο	**n(2m+1)- m(2m-1) +1**	**m(2n+3) +1**

Από τον παραπάνω πίνακα προκύπτει ότι η αναθεωρημένη μορφή εκτελεί ελάχιστα περισσότερες πράξεις. Έστω ότι θέλουμε να λύσουμε ένα γραμμικό πρόβλημα 100 x 200 το οποίο είναι στη τυποποιημένη μορφή. Κλασική μέθοδος θα εκτελέσει 20301 πράξεις ενώ η αναθεωρημένη 40301. Τα περισσότερα όμως γραμμικά προβλήματα είναι πολύ αραιά (πυκνότητα μικρότερη του 5%) με αποτέλεσμα ο αναθεωρημένος αλγόριθμος να μπορεί να αποφύγει πράξεις με μηδενικά. Στη συνέχεια παρατίθεται ο πίνακας με την πολυπλοκότητα των αλγορίθμων.

	Κλασική	**Αναθεωρημένη**
Μνήμη	$O(mn)$	$O(m^2)$
Καλύτερη περίπτωση	$O(mn)$	$O(m^2)$
Χειρότερη περίπτωση	$O(mn)$	$O(nm)$

3.2 Εύρεση εφικτής αρχικής βάσης

Στην προηγούμενη παράγραφο θεωρήθηκε ότι για το γραμμικό πρόβλημα που επιλύεται υπάρχει μια γνωστή εφικτή βασική λύση. Όπως όμως αναφέρθηκε στο γενικό γραμμικό πρόβλημα μια τέτοια λύση δεν είναι γνωστή. Για να βρεθεί μια αρχική εφικτή λύση έχουν αναπτυχθεί δύο μεθοδολογίες, η μέθοδος των δύο φάσεων (Two Phase method) και η μέθοδος του μεγάλου M (Big-M method). Στην υλοποίηση του αλγορίθμου χρησιμοποιήθηκε η μέθοδος των δύο φάσεων. Πριν όμως από αυτήν πρέπει να εκτελεστεί η διαδικασία lprref η οποία δημιουργεί την βασική διαμέριση της μήτρας A σε περίπτωση που το γραμμικό πρόβλημα περιλαμβάνει και ισοτικούς περιορισμούς εξ' αρχής. Σε περίπτωση που η βάση B η οποία επιλέγεται είναι εφικτή τότε η μέθοδος των δύο φάσεων παραλείπεται, σε περίπτωση όμως που από την διαμέριση προκύπτει μη εφικτή βάση τότε εκτελείται η μέθοδος των δύο φάσεων με μια τεχνητή μεταβλητή (two phase method with one artificial variable).

3.2.1 lprref

Η διαδικασία lprref λαμβάνει ως είσοδο την μήτρα A και επιστρέφει μια βασική διαμέριση (B N). Δηλαδή μια βάση B η οποία είναι αντιστρέψιμη. Για να γίνει αυτό εκτελούνται γραμμοπράξεις στη μήτρα A μέχρι να σχηματιστεί ο μοναδιαίος πίνακας. Μετά το πέρας αυτής της διαδικασίας η μοναδιαία μήτρα είναι η βάση B ενώ οι υπόλοιπες στήλες συνθέτουν τον πίνακα N.

Σε ότι αφορά την υλοποίηση, επειδή η διαδικασία αυτή εκτελείται μετά την προσθήκη χαλαρών μεταβλητών υπάρχουν ήδη δημιουργημένες κάποιες μοναδιαίες στήλες. Κατά την διαδικασία addSlackVar

επιστρέφεται μια λίστα στην οποία εμφανίζονται οι γραμμές στις οποίες πρέπει να δημιουργηθούν μοναδιαίες στήλες. Στη συνέχεια το στοιχείο περιστροφής αναζητείται μόνο σε αυτές τις γραμμές ώστε να μην δημιουργηθεί ξανά μια ήδη υπάρχουσα μοναδιαία στήλη. Για κάθε στήλη επιλέγεται ως στοιχείο περιστροφής το μεγαλύτερο κατά απόλυτη τιμή στοιχείο, έχουμε δηλαδή πλήρη οδήγηση. Σε περίπτωση που το στοιχείο αυτό βρίσκεται σε γραμμή διαφορετική από αυτή που χρειάζεται να είναι εκτελείται εναλλαγή γραμμών.

Στο τέλος της ρουτίνας, γίνεται έλεγχος για το αν η βάση που επιλέχθηκε είναι εφικτή. Η συνάρτηση επιστρέφει την τιμή 1 αν η εκτέλεση του πρόβλημα θα συνεχιστεί με την μέθοδο των δύο φάσεων και την τιμή 2 αν θα εκτελεστεί απ' ευθείας ο αναθεωρημένος αλγόριθμος simplex όπως παρουσιάστηκε στη παράγραφο 3.1.1.

Παρακάτω δίνεται ο κώδικας για τη δημιουργία μοναδιαίου πίνακα με πλήρη οδήγηση και ο έλεγχος για το αν η βάση είναι εφικτή.

```
//-------------------------Create singleton-------------
   long double pivot;
   j=0;
   int k,m;
   int lem;
   while(1) {
      pos1=equals->getNth(1);
      if (pos1==0) break;

      long double max=0;
      int maxi=-1;
      while (1) {
        if (fabs(max)>tol) {j--; break;}
        for (i=0;i<rows;i++) {
           if (equals->find(i+1)==0) continue;
           value=a[i][j];
            if (fabs(a[i][j])>fabs(max)) {
                  max=a[i][j];
                  maxi=i;
            }
        }
        j++;
        if (j==cols1) return(-1);
      }

      i=maxi;
      b_index[pos1-1]=j;

      pivot=max;
      rhs[i]=rhs[i]/pivot;
      for (m=0;m<cols;m++) {
         value=a[i][m]/pivot;
         if (fabs(value)>tol) a[i][m]=value;
            else a[i][m]=0;
      }
```

```
            for (k=0;k<rows;k++) {
                 if (a[k][j]!=0) {
                       long double pivot1=a[k][j];
                       if (k!=i) {
                       value=rhs[k]-rhs[i]*pivot1;
                  if (fabs(value)>tol) rhs[k]=value;
                     else rhs[k]=0;
             }
             for (m=0;m<cols;m++) {
                if (k!=i) {
                  value=a[k][m]-(pivot1*a[i][m]);
                     if (fabs(value)>tol) a[k][m]=value;
                        else a[k][m]=0;
                }
             }
          }
}
if (i!=pos1-1) {
         value=rhs[i];
         rhs[i]=rhs[pos1-1];
         rhs[pos1-1]=value;
         for (int lem=0;lem<cols;lem++) {
             value=a[i][lem];
             a[i][lem]=a[pos1-1][lem];
             a[pos1-1][lem]=value;
         }
```

Έλεγχος δύο φάσεων:

```
   bool feasible=true;
   for (i=0;i<rows;i++) {
      value=b[i][i]*rhs[i];
      if (abs(value)<tol) value=0;
      if (value<0) {
         feasible=false;
         break;
      }
      if (feasible==false) break;
   }
   e=getCPUTime();
   lprrefTime=(e-s);
   if (feasible==true) return 2;
      else return 1;
```

3.2.2 Μέθοδος δύο φάσεων

Έστω ότι θέλουμε να επιλύσουμε το γενικό γραμμικό πρόβλημα με τον αναθεωρημένο αλγόριθμο simplex. Σε περίπτωση που από την βασική διαμέριση που επιστράφηκε από το lprref δεν βρέθηκε ένα αρχικό βασικό εφικτό σημείο $x^T = (x_B, x_N) \geq 0$, πρέπει να χρησιμοποιηθεί μια μέθοδος για

τον υπολογισμό του. Μια τέτοια μέθοδος είναι των δύο φάσεων με μια τεχνητή μεταβλητή (two phase variable with one artificial variable). Σε αυτήν ο αλγόριθμος simplex εφαρμόζεται σε δύο φάσεις, στη φάση I επιλύεται ένα τροποποιημένο γραμμικό πρόβλημα, ενώ το αρχικό γραμμικό πρόβλημα επιλύεται στη φάση II αφού έχει ήδη βρεθεί ένα αρχικό βασικό εφικτό σημείο από τη φάση I.

Το τροποποιημένο πρόβλημα της φάσης I κατασκευάζεται ως εξής:

- Εισάγεται μια νέα μεταβλητή x_{n+1} η οποία ονομάζεται τεχνητή (artificial) με συντελεστές d = - Be με e ένα διάνυσμα διαστάσεων 1 x m με όλες τις τιμές του μονάδες.
- Αντικαθιστούμε την αντικειμενική συνάρτηση με την min x_{n+1}

Έτσι το γραμμικό πρόβλημα έχει τη μορφή:

$$\begin{aligned} &min && x_{n+1} \\ &\mu.\pi. && Ax + dx_{n+1} = b \qquad (\Gamma.\Pi.\ 4) \\ &x,\ x_{n+1} \geq 0 \end{aligned}$$

```
d=(long double *) new long double[rowsAfter];
h=(long double *) new long double[rowsAfter];
f=(long double *) new long double[columnsAfter+1];
xb=(long double *) new long double[rowsAfter];
wt=(long double *) new long double[rowsAfter];
sn=(long double *) new long double[columnsAfter-rowsAfter+1];
n_index_new=(int *) new int[columnsAfter-rowsAfter+1];
n_new=(long double **) new long double*[rowsAfter];

for (i=0;i<rowsAfter;i++)
   n_new[i]=(long double*) new long double[columnsAfter-rowsAfter+1];
for (j=0;j<columnsAfter-rowsAfter;j++)
     n_index_new[j]=n_index[j];
n_index_new[columnsAfter-rowsAfter]=columnsAfter;
for (i=0;i<rowsAfter;i++)
      for (j=0;j<columnsAfter-rowsAfter;j++)   n_new[i][j]=n[i][j];
//-----------Create d,N,B,f-----------------------------------
for (i=0;i<rowsAfter;i++) {
     d[i]=-b[i][i]; h[i]=b[i][i]*d[i];
}
for (i=0;i<rowsAfter;i++)
     n_new[i][columnsAfter-rowsAfter]=d[i];
for (j=0;j<columnsAfter;j++) f[j]=0;
f[columnsAfter]=1;
```

Στη συνέχεια πρέπει να γίνει μια περιστροφή ώστε η τεχνητή μεταβλητή να εισέρθει στη βάση. Ως εξερχόμενη μεταβλητή επιλέγεται

εκείνη με τη μικρότερη τιμή δεξιού μέρους και ανανεώνονται τα σύνολα δεικτών σύμφωνα με τη σχέση:

$$Bnew \leftarrow Bold \setminus \{k\} \cup \{n+1\}$$
$$Nnew \leftarrow Nold \setminus \{n+1\} \cup \{k\}$$

Η στήλη περιστροφής είναι αυτή που αντιστοιχεί στη τεχνητή μεταβλητή και υπολογίζεται από την ακόλουθη σχέση:

$$h_{n+1} = B^{-1}A_{.n+1}$$

Τέλος, χρησιμοποιώντας την νέα βασική διαμέριση (B N) υπολογίζουμε τις νέες μήτρες B^{-1}, x_B και s_N. Η επιλογή της (n+1) μεταβλητής ως εισερχόμενης και της k ως εξερχόμενης εγγυάται την κατασκευή ενός βασικού εφικτού σημείου για το τροποποιημένο πρόβλήμα. Έτσι, τώρα μπορεί να εκτελεστεί ο αλγόριθμος simplex. Η μόνη διαφορά σε σχέση με τη διαδικασία που αναφέρθηκε στη παράγραφο 3.1.1 έγκειται στο πότε ο αλγόριθμος σταματάει και γίνεται η μετάβαση στη φάση II.

Η φάση I λοιπόν τερματίζεται με δύο τρόπους, είτε με την έξοδο της τεχνητής μεταβλητής από τη βάση είτε με την εύρεση βέλτιστης λύσης στο πρόβλημα. Στη πρώτη περίπτωση το πρόγραμμα χωρίς καμία αλλαγή είναι έτοιμο για μετάβαση στη φάση II με βασική διαμέριση (B N) όπου N ο δείκτης N χωρίς τον δείκτη της τεχνητής μεταβλητής.

```
  if (exported==columnsAfter) {
          b_index[exported_b_index]=imported;
          n_index_new[imported_n_index]=exported;
          for (i=0;i<rowsAfter;i++) d[i]=b[i][exported_b_index];
          for (i=0;i<rowsAfter;i++)
                b[i][exported_b_index]=n_new[i][imported_n_index];
          for (i=0;i<rowsAfter;i++) n_new[i][imported_n_index]=d[i];
          int k=0;
          int m;
          for (i=0;i<columnsAfter-rowsAfter+1;i++) {
                  if (n_index_new[i]!=columnsAfter)
                                n_index[i-k]=n_index_new[i];
                  else {k++; m=i;}
          }
          for (i=0;i<rowsAfter;i++) {
                  for (j=0;j<columnsAfter-rowsAfter+1;j++) {
                      if (j<m)  n[i][j]=n_new[i][j];
                            else if (j>m) n[i][j-1]=n_new[i][j];
                  }
          }
          e=getCPUTime();
          algTime=(e-s);
return(phase2(tol,binv,h,exported_b_index,rowsAfter,columnsAfter,a,
Anz_C,JA_C,b,b_index,n,n_index,rhs,algTime,z,iter));
```

Στη δεύτερη περίπτωση απαιτείται ένας επιπλέον έλεγχος. Αν για το x_{n+1} το οποίο βρίσκεται μέσα στη βάση ισχύει $x_{n+1}>0$ τότε το πρόβλημα είναι αδύνατο. Σε αντίθετη περίπτωση ($x_{n+1} = 0$) τότε εκτελείται άλλη μια περιστροφή για να βγει η τεχνητή μεταβλητή από τη βάση και η νέα διαμέριση (B N) περνάει στη φάση II.

```
int m;
for (i=0;i<rowsAfter;i++) {
      if (b_index[i]==columnsAfter) {stop=true; m=i; break;}
}
if (stop==true) {     // artificial variable in B
      if (xb[m]>0) {e=getCPUTime(); algTime=(e-s);return(1);}
         else {
             int k;
             int r;
             r=m;
             exported_b_index=r;
             exported=b_index[exported_b_index];
```

```
        for (i=0;i<columnsAfter-rowsAfter+1;i++) {
           for (j=0;j<rowsAfter;j++) d[j]=a[j][imported];
           for (j=0;j<rowsAfter;j++) {
                  value=0;
                  for (k=0;k<rowsAfter;k++) {
                        if (binv[j][k]!=0 && fabs(binv[j][k])>tol
&& d[k]!=0 && fabs(d[k])>tol)
                           value=value+(binv[j][k]*d[k]);
                        }
                  if (fabs(value)>tol) h[j]=value;
                              else h[j]=0;
                  }
                  if (h[r]!=0) {
                        imported_n_index=i;
                        imported=n_index_new[imported_n_index];
                        break;
                  }
                  b_index[exported_b_index]=imported;
                  n_index_new[imported_n_index]=exported;
            }
            for (i=0;i<rowsAfter;i++) d[i]=b[i][exported_b_index];
            for (i=0;i<rowsAfter;i++)
                 b[i][exported_b_index]=n_new[i][imported_n_index];
            for (i=0;i<rowsAfter;i++)
                       n_new[i][imported_n_index]=d[i];
            k=0;
            for (i=0;i<columnsAfter-rowsAfter+1;i++) {
                  if (n_index_new[i]!=columnsAfter)
                              n_index[i-k]=n_index_new[i];
                  else {k++; m=i;}
            }
```

```
            for (i=0;i<rowsAfter;i++) {
                for (j=0;j<columnsAfter-rowsAfter+1;j++) {
                    if (j<m)  n[i][j]=n_new[i][j];
                          else if (j>m) n[i][j-1]=n_new[i][j];
                    }
            }
            e=getCPUTime();
            algTime=(e-s);

return(phase2(tol,binv,h,exported_b_index,rowsAfter,columnsAfte
r,a,Anz_C,JA_C,b,b_index,n,n_index,rhs,algTime,z,iter));
}
```

Η διαδικασία που αναφέρθηκε προηγούμενα συνοψίζεται στα παρακάτω βήματα:

Μέθοδος δύο φάσεων

1. *Έλεγχος εφικτότητας. Ξεκίνα με μια βασική διαμέριση (B N). Υπολόγισε τη μήτρα B^{-1} και τα διανύσματα x_B, s_N. Αν ισχύει $x_B = B^{-1}b \geq 0$ πήγαινε στο βήμα 4. Διαφορετικά υπολόγισε τους συντελεστές της μεταβλητής x_{n+1}.*
2. *Περιστροφή. Επέλεξε ως εισερχόμενη τη μεταβλητή x_{n+1} και ως εξερχόμενη αυτή με το μικρότερο δεξιό μέρος. Ανανέωσε τα σύνολα των δεικτών B και N. Υπολόγισε τη στήλη περιστροφής και ανανέωσε την αντίστροφη της βάσης B και τα διανύσματα x_B, s_N.*
3. *Φάση Ι. Χρησιμοποίησε τη νέα βασική διαμέριση και εφάρμοσε τον αναθεωρημένο αλγόριθμο simplex στο τροποποιημένο πρόβλημα. Αν η τεχνητή μεταβλητή βγει από τη βάση πήγαινε το βήμα 4, αν βρεθεί βέλτιστη λύση, αν $x_{n+1}>0$ τότε τερματίζεται ο αλγόριθμος το πρόβλημα είναι αδύνατο, διαφορετικά εκτέλεσε μια περιστροφή για να βγει η τεχνητή μεταβλητή από τη βάση και πήγαινε στο βήμα 4*
4. *Φάση ΙΙ. Χρησιμοποιώντας τη βασική διαμέριση που προέκυψε από τη φάση Ι εκτέλεσε τον αναθεωρημένο αλγόριθμο simplex για το αρχικό γραμμικό πρόβλημα.*

Παράδειγμα

Να λυθεί το γραμμικό πρόβλημα:

min x2 + 10x3

μ.π. -2x1 – x2 + 4x3 + x4 = - 4

3x1 + x2 - x3 - x5 = 5

$x_j \geq 0$, j = 1, 2,…, 5

Ξεκινάμε με B = [4 5] και

$B^{-1} = \begin{matrix} 1 & 0 \\ 0 & -1 \end{matrix}$

$x_B = B^{-1}b = \begin{matrix} -4 \\ -5 \end{matrix}$

Η λύση αυτή δεν είναι εφικτή, οπότε καταφεύγουμε στη μέθοδο των δύο φάσεων. min {x_B} = --5 άρα r = 2.

$d = -Be = \begin{matrix} -1 \\ 1 \end{matrix}$

Άρα το τροποποιημένο πρόβλημα είναι της μορφής:

min x6
μ.π. -2x1 – x2 + 4x3 + x4 - x6 = - 4
3x1 + x2 - x3 - x5 + x6 = 5

Φάση Ι – Επανάληψη 1

B = [4 6] N = [1 2 3 5]

$(f_B)^T = (0\ 1)$ $(f_N)^T = (0\ 0\ 0\ 0)$

$B = \begin{matrix} 1 & -1 \\ 0 & 1 \end{matrix}$ $N = \begin{matrix} -2 & -1 & 4 & 0 \\ 3 & 1 & -1 & -1 \end{matrix}$

$B^{-1} = \begin{matrix} 1 & 1 \\ 0 & 1 \end{matrix}$

$x_B = B^{-1} b = \begin{matrix} 1 \\ 5 \end{matrix}$

$w^T = (f_B)^T B^{-1} = [0\ 1]$

$s_N = (f_N)^T – w^T N = [-3\ -1\ 1\ 1]$

Επειδή, $s_N \geq 0$ ο αλγόριθμος δεν σταματά και σύμφωνα με τον κανόνα του Dantzig επιλέγεται η εξερχόμενη μεταβλητή l=1και t=1.

Υπολογίζουμε το διάνυσμα hl

$h_l = B^{-1}A_{.l} = \begin{matrix} 1 \\ 3 \end{matrix}$

Επειδή, $h_l \leq 0$ ο αλγόριθμος δεν σταματά και σύμφωνα με το τέστ ελαχίστου λόγου επιλέγεται η εξερχόμενη μεταβλητή.

$\min \{ x_b[1] / h_{l1}, x_b[2] / h_{l2}\} = \min (1, 5/3) = 1$

Επομένως, r=1 και k=4.

Εκτελούμε την περιστροφή οπότε, B = [1 6] και N [4 2 3 5]

Φάση I – Επανάληψη 2

$(f_B)^T = (0\ 1)$ $(f_N)^T = (0\ 0\ 0\ 0)$

$B = \begin{matrix} -2 & -1 \\ 3 & 1 \end{matrix}$ $N = \begin{matrix} 1 & -1 & 4 & 0 \\ 0 & 1 & -1 & -1 \end{matrix}$

$B^{-1} = \begin{matrix} 1 & 1 \\ -3 & 2 \end{matrix}$

$x_B = B^{-1} b = \begin{matrix} 1 \\ 2 \end{matrix}$

$w^T = (f_B)^T B^{-1} = [-3\ -2]$

$s_N = (f_N)^T - w^T N = [3\ -1\ 10\ -2]$

Επειδή, $s_N \geq 0$ ο αλγόριθμος δεν σταματά και σύμφωνα με τον κανόνα του Dantzig επιλέγεται η εξερχόμενη μεταβλητή l=5και t=4.

Υπολογίζουμε το διάνυσμα hl

$h_l = B^{-1} A_{.l} = \begin{matrix} -1 \\ 2 \end{matrix}$

Επειδή, $h_l \leq 0$ ο αλγόριθμος δεν σταματά και σύμφωνα με το τέστ ελαχίστου λόγου επιλέγεται η εξερχόμενη μεταβλητή. Επειδή όμως υπάρχει μόνο μία τιμή θετική r=2 και k=6. Εξερχόμενη μεταβλητή είναι η τεχνητή, άρα περνάμε στη φάση II.

Φάση II – Επανάληψη 3

$B = [1\ 5] \quad N = [4\ 2\ 3]$

$(c_B)^T = (0\ 0) \quad (c_N)^T = (0\ 1\ 10)$

$$B = \begin{matrix} -2 & 0 \\ 3 & 1 \end{matrix} \qquad N = \begin{matrix} 1 & -1 & 4 \\ 0 & 1 & -1 \end{matrix}$$

$$B^{-1} = \begin{matrix} -1/2 & 0 \\ -3/2 & -2 \end{matrix}$$

$$x_B = B^{-1} b = \begin{matrix} 2 \\ 1 \end{matrix}$$

$w^T = (c_B)^T B^{-1} = [0\ 0]$

$s_N = (c_N)^T - w^T N = [0\ 1\ 10]$

Επειδή $s_N \geq 0$ *η τρέχουσα λύση είναι η βέλτιστη και ο αλγόριθμος σταματάει.*

3.2.3 Μέθοδος μεγάλου Μ

Στη μέθοδο του μεγάλου Μ δεν λύνονται δύο διαφορετικά προβλήματα αλλά ένα τροποποιημένο πρόβλημα, από τη λύση του οποίου εξάγονται συμπεράσματα για τη λύση του αρχικού προβλήματος. Το τροποποιημένο πρόβλημα, που ονομάζεται πρόβλημα του μεγάλου Μ (big M problem), έχει τις ίδιες μεταβλητές και τους ίδιους περιορισμούς με το πρόβλημα της φάσης Ι. Η μόνη διαφορά βρίσκεται στην αντικειμενική συνάρτηση, η οποία αποτελείται από δύο όρους. Ο ένας είναι η αντικειμενική συνάρτηση του αρχικού προβλήματος και ο άλλος είναι η τεχνητή μεταβλητή πολλαπλασιασμένη με έναν πάρα πολύ μεγάλο αριθμό Μ.

$$\begin{aligned} &min \quad c^T x + M x_{n+1} \\ &\mu.\pi. \quad Ax + d x_{n+1} = b \\ &x_{n+1} \geq 0 \end{aligned}$$

Η τιμή του Μ μπορεί να υπολογιστεί από τα δεδομένα του προβλήματος. Συνηθισμένες τιμές του Μ στην πράξη είναι $M = 10^{20}$ ή $M = 10^{30}$. Αν το πρόβλημα του μεγάλου Μ είναι απεριόριστο τότε το αρχικό πρόβλημα είναι ή αδύνατο ή απεριόριστο. Αν το πρόβλημα είναι

βέλτιστο, τότε το αρχικό πρόβλημα είναι βέλτιστο ή αδύνατο. Πιο συγκεκριμένα, αν η τιμή της τεχνητής μεταβλητής είναι μηδέν τότε το αρχικό πρόβλημα είναι βέλτιστο.

Παράδειγμα
Να λυθεί το γραμμικό πρόβλημα με τη μέθοδο του μεγάλου Μ:

min x2 + 10x3
μ.π. -2x1 – x2 + 4x3 + x4 = - 4
3x1 + x2 - x3 - x5 = 5
$x_j \geq 0$, j = 1, 2,…, 5

Το πρόβλημα για Μ=100 το πρόβλημα παίρνει την μορφή:

min x2 + 10x3 + 100x6
μ.π. -2x1 – x2 + 4x3 + x4 - x6 = - 4
3x1 + x2 - x3 - x5 + x6 = 5
$x_j \geq 0$, j = 1, 2,…, 6

Επανάληψη 1

Η πρώτη εφικτή βάση κατασκευάζεται όπως στη φάση Ι, δηλαδή Β = [4 6] και Ν = [1 2 3 5]

$(g_B)^T$ = [0 100] $(g_N)^T$ = [0 1 10 0]
w^T = [0 100] 1 1 = [0 100]
0 1
$(s_N)^T$= [-300 -99 110 100]
Εφαρμόζοντας τον κανόνα του Dantzig βρίσκουμε ότι εισερχόμενος είναι ο l = 1 και t =1 και εξερχόμενη η r=1 και k=4.

Επανάληψη 2

Η πρώτη εφικτή βάση κατασκευάζεται όπως στη φάση Ι, δηλαδή Β = [1 6] και Ν = [4 2 3 5]

$(g_B)^T$ = [0 100] $(g_N)^T$ = [0 1 10 0]
w^T = [-300 -200]
$(s_N)^T$= [300 -99 1010 -200]
Εισερχόμενη είναι η μεταβλητή 5 και εξερχόμενη η μεταβλητή 6.

Επανάληψη 3

Η πρώτη εφικτή βάση κατασκευάζεται όπως στη φάση I, δηλαδή B = [1 5] και N = [4 2 3 6]

$(g_B)^T$ *= [0 0]* $(g_N)^T$ *= [0 1 10 100]*
w^T *= [0 0]*
$(s_N)^T$*= [0 1 10 100]*

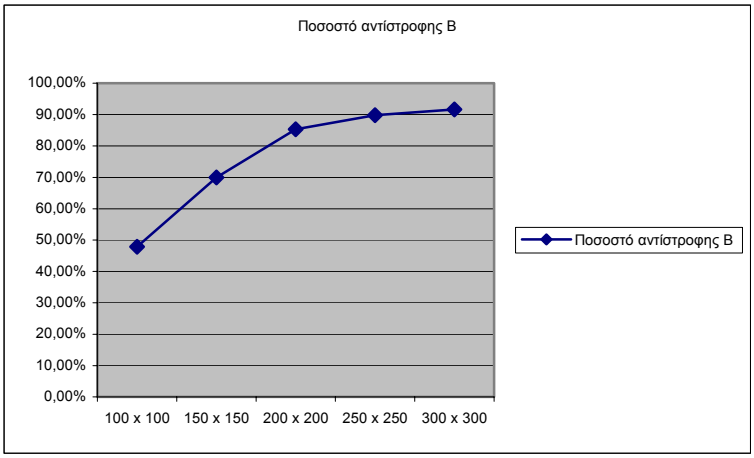
Επειδή $s_N \geq 0$ *το τρέχον σημείο είναι βέλτιστο και οι υπολογισμοί σταματούν. Επιπλέον, επειδή η τιμή της τεχνητής μεταβλητής στη βέλτιστη λύση είναι μηδέν συμπεραίνουμε ότι το αρχικό πρόβλημα είναι βέλτιστο.*

3.3 Αντιστροφή μήτρας

Η διαδικασία αντιστροφής της βάσης B^{-1} είναι το πιο χρονοβόρο υπολογιστικά βήμα στους αλγορίθμους τύπου Simplex. Πιο συγκεκριμένα, καταλαμβάνει το 70% με 80% του συνολικό χρόνο.

Διαστάσεις (1)	**nnz (2)**	**Χρόνος B^{-1} (3)**	**Συνολικός χρόνος (4)**	**Λόγος (3) / (4)**
100 x 100	488	0.161	0.351	47.87 %
150 x 150	1097	5.123	7.317	70.01 %
200 x 200	1951	26.798	31.348	85.38 %
250 x 250	3045	83.793	93.291	89.82 %
300 x 300	4386	232.523	253.668	91.66 %

Από τον παραπάνω πίνακα παρατηρούμε ότι όσο αυξάνεται η διάσταση του γραμμικού προβλήματος τόσο περισσότερο χρόνο καταλαμβάνει ο υπολογισμός του B^{-1} στο συνολικό χρόνο επίλυσης.



Για τη δημιουργία της παραπάνω έρευνας χρησιμοποιήθηκε η ενσωματωμένη στο Matlab συνάρτηση inv(erse). Η συνάρτηση αυτή όπως έχει ήδη αναφερθεί έχει αντικατασταθεί από τη χρήση δύο μεθόδων, την eta-matrix αντιστροφή και την rank-one, οι οποίες βασίζονται στον υπολογισμό της νέας αντίστροφης από την προηγούμενη. Όμως παρά την σημαντική μείωση στο χρόνο αντιστροφής με αυτές τις μεθόδους σε ορισμένα προβλήματα πρέπει να χρησιμοποιηθεί και ο υπολογισμός της αντίστροφης B^{-1} από τη βάση B γιατί η ανανέωση της προκαλεί

υπολογιστικά λάθη. Αυτά μπορούν να οδηγήσουν σε αλλοίωση της βέλτιστης λύσης ή ακόμα και σε αδυναμία επίλυσης του προβλήματος.

Στην υλοποίηση του αναθεωρημένου αλγόριθμου simplex χρησιμοποιήθηκε η συνάρτηση full_Inverse η οποία υπολογίζει την αντίστροφη μια βάσης με την μέθοδο Gauss με πλήρη οδήγηση [6].

```
for (i=0;i<n;i++) {
   for (j=0;j<n;j++) {
       a1[i][j]=a[i][j];
       if (i==j) binv1[i][j]=1;
          else binv1[i][j]=0;
   }
 }
 for (j=0;j<n;j++) {
    max=0;
    ip=-1;
    for (i=j;i<n;i++) {
      if (fabs(a1[i][j])>fabs(max)) {
        max=a1[i][j];
        ip=i;
      }
    }
    if (max==0) cout<<"Matrix cannot be inversed"<<endl;
    if (ip!=j) {
       for (i=0;i<n;i++) {
          max=a1[ip][i];
          a1[ip][i]=a1[j][i];
          a1[j][i]=max;
          max=binv1[ip][i];
          binv1[ip][i]=binv1[j][i];
          binv1[j][i]=max;
       }
    }
    pivot=a1[j][j];
    for (i=0;i<n;i++) {
      a1[j][i]=a1[j][i]/pivot;
      binv1[j][i]=binv1[j][i]/pivot;
    }
    for (i=0;i<n;i++) {
      pivot1=a1[i][j];
      if (pivot1==0) continue;
      for (ip=0;ip<n;ip++) {
         if (i!=j) {
           a1[i][ip]=a1[i][ip]-a1[j][ip]*pivot1;
           binv1[i][ip]=binv1[i][ip]-binv1[j][ip]*pivot1;
         }
      }
    }
 }
 for (i=0;i<n;i++)
   delete a1[i];
 delete a1;
```

3.3.1 E-matrix inversion

Μια μήτρα Ε ονομάζεται στοιχειώδης (elementary) αν διαφέρει από την μοναδιαία μήτρα I κατά μια μόνο στήλη. Έστω ότι η μήτρα Ε διαφέρει από τη μοναδιαία στη στήλη j, τότε συμβολίζεται ως εξής:

$$E = (e_1, e_2, \dots, e_{j-1}, a, e_{j+1}, \dots, e_{m-1}. e_m)$$

ή

$$E = \begin{pmatrix} 1 & \dots & a1 & \dots & 0 \\ 0 & . & . & & 0 \\ & . & . & & \\ & & a_{ij} & & \\ & & . & . & \\ 0 & & . & & . \\ 0 & & a_m & & 1 \end{pmatrix}$$

Η στοιχειώδης μήτρα Ε ονομάζεται και μήτρα Frobenius. Η μήτρα Ε γράφεται επίσης και ως:

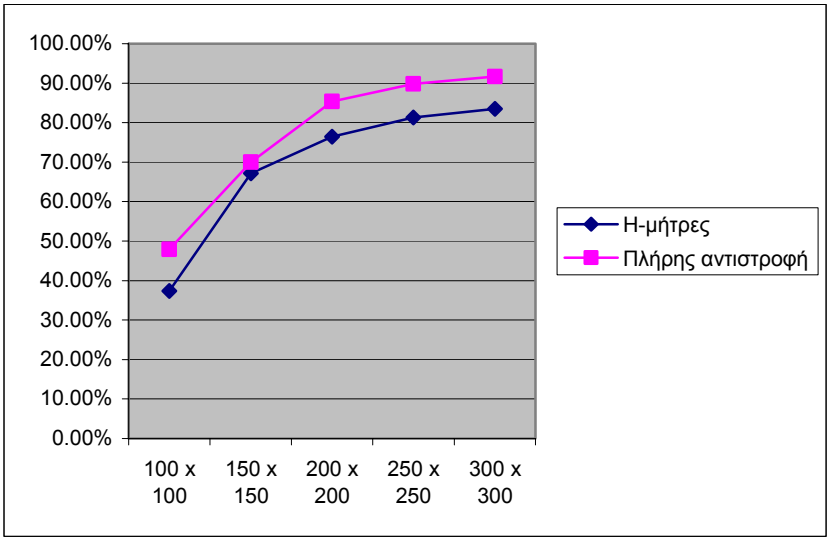
$$E = I + (a - e_j)\, e_j^T$$

Αποδεικνύεται ότι η αντίστροφη της μήτρας Ε είναι η μοναδιαία αντικαθιστώντας τη στήλη j με $E^{-1}[i][j] = -aij / ajj$ για $j \neq i$ και $E^{-1}[j][j] = 1/ ajj$. Η νέα αντίστροφη $B^{-1}(new)$ της Β σε κάθε επανάληψη του αλγορίθμου μπορεί να υπολογιστεί από την $B^{-1}(old)$ με μία απλή πράξη περιστροφής. Έχουμε δηλαδή,

$$B^{-1}(new) = E^{-1}B^{-1}(old)$$

Με τη μέθοδο αυτή το ποσοστό συμμετοχής του υπολογισμό αντίστροφής στο συνολικό χρόνο επίλυσης μειώνεται σε 60% με 70%.

Διαστάσεις (1)	**nnz (2)**	**Χρόνος B^{-1} (3)**	**Συνολικός χρόνος (4)**	**Λόγος (3) / (4)**
100 x 100	488	0.129	0.346	37.37%
150 x 150	1097	4.455	6.631	67.18%
200 x 200	1951	18.064	23.632	76.44%
250 x 250	3045	72.782	89.514	81.30%
300 x 300	4386	136.618	163.548	83.53%



Παράδειγμα

Έστω $B^{-1}(old) = \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$

η στήλη που θα εισαχθεί είναι a2= $\begin{matrix} -1 \\ 2 \\ -2 \end{matrix}$

και θα εισαχθεί στη στήλη 2.

Τότε, $h2 = \begin{matrix} -(-1)/2 \\ 1/2 \\ -(-2)/2 \end{matrix} = \begin{matrix} 1/2 \\ 1/2 \\ 1 \end{matrix}$

$E^{-1} = \begin{matrix} 1 & 0.5 & 0 \\ 0 & 0.5 & 0 \\ 0 & 1 & 1 \end{matrix}$

$B^{-1}(new) = E^{-1} B^{-1}(old) = \begin{matrix} 1 & 0.5 & 0 \\ 0 & 0.5 & 0 \\ 0 & 1 & 1 \end{matrix} * \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix} \rightarrow$

$B^{-1}(new) = \begin{matrix} 1 & 0.5 & 0 \\ 0 & 0.5 & 0 \\ 0 & 1 & 1 \end{matrix}$

Όπως εύκολα μπορεί να αποδειχθεί ο πίνακας αυτός είναι ο αντίστροφος του

B (new) = 1 -1 0
0 2 0
0 -2 1

Ο κώδικας που εκτελεί την παραπάνω διαδικασία είναι ο εξής:

```
 void etaInverse(long double **a1,long double* h1,int pos,int n) {

long double ** eta, **y;
int i,j; long double value;

eta=(long double **) new long double*[n];
   for (i=0;i<n;i++) eta[i]=(long double*) new long double[n];
   y=(long double **) new long double*[n];
   for (i=0;i<n;i++) y[i]=(long double*) new long double[n];
   value=h1[pos];
   for (i=0;i<n;i++)
      if (i==pos) h1[i]=1/h1[i];
         else h1[i]=-h1[i]/value;
   for (j=0;j<n;j++) {
      if (j!=pos)
           for (i=0;i<n;i++) if (i==j) eta[i][j]=1; else
eta[i][j]=0;
        else
           for (i=0;i<n;i++) eta[i][j]=h1[i];
   }
   for (i=0;i<n;i++) {
      for (j=0;j<n;j++) {
         value=0;
         value=value+eta[i][i]*a1[i][j];
         if (i!=pos) value=value+eta[i][pos]*a1[pos][j];
         y[i][j]=value;
      }
   }
```

```
   for (i=0;i<n;i++)
     for (j=0;j<n;j++)
       a1[i][j]=y[i][j];

   for (i=0;i<n;i++) {
      delete eta[i];
      delete y[i];
   }
   delete y;
   delete eta;
}
```

3.3.2 Rank one inversion

Η μέθοδος με τις Ε μήτρες, μπορεί να είναι αρκετά πιο γρήγορη από την πλήρη αντιστροφή με τη μέθοδο Gauss, αλλά η πολυπλοκότητα της είναι $O(m^3)$ λόγω του πολλαπλασιασμού πινάκων που περιλαμβάνει. Παρόμοια με τη μέθοδο αυτή λειτουργεί και η μέθοδος Rank one με πολυπλοκότητα $O(m^2)$ επειδή περιλαμβάνει πράξη εξωτερικού γινόμενου διανυσμάτων αντί για πολλαπλασιασμό μητρών.

Πιο συγκεκριμένα, υπολογίζει δύο mxm μήτρες τις οποίες στη συνέχεια προσθέτει. Η πρώτη μήτρα προκύπτει από τον B^{-1}(old) αντικαθιστώντας με μηδενικά τη γραμμή που είναι εξερχόμενη από τον πίνακα Β. Η δεύτερη μήτρα προκύπτει από το εξωτερικού γινόμενο της στήλης h, όπως προκύπτει από τη στήλη που θα εισαχθεί στο Β μετά από περιστροφή, και της γραμμής του B^{-1}(old) που αντιστοιχεί στην εξερχόμενη μεταβλητή.

Παράδειγμα

Έστω $B^{-1}(old)$ $= \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$

η στήλη που θα εισαχθεί είναι a2= $\begin{matrix} -1 \\ 2 \\ -2 \end{matrix}$

και θα εισαχθεί στη στήλη 2.

Τότε, h2 $= \begin{matrix} -(-1)/2 \\ 1/2 \\ -(-2)/2 \end{matrix} = \begin{matrix} 1/2 \\ 1/2 \\ 1 \end{matrix}$

Εξωτερικό γινόμενο K $= \begin{matrix} 0.5 \\ 0.5 \\ 1 \end{matrix} [0\ 1\ 0] = \begin{matrix} 0 & 0.5 & 0 \\ 0 & 0.5 & 0 \\ 0 & 1 & 0 \end{matrix}$

και τροποποιημένη μήτρα

B^{-1} *(old)* $= \begin{matrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{matrix}$

οπότε ,

$B^{-1}(new) = B^{-1}\ (old) + K =$

$$= \begin{matrix} 1 & 0.5 & 0 \\ 0 & 0.5 & 0 \\ 0 & 1 & 1 \end{matrix}$$

Όπως εύκολα μπορεί να αποδειχθεί ο πίνακας αυτός είναι ο αντίστροφος του B (new) = $\begin{matrix} 1 & -1 & 0 \\ 0 & 2 & 0 \\ 0 & -2 & 1 \end{matrix}$

Ο κώδικας που εκτελεί την παραπάνω διαδικασία είναι ο εξής:

```
void rankOneInverse(long double **a1,long double*
h1,int pos,int n) {

long double **b_ch;
long double **y;
int i,j;
long double value;

   b_ch=(long double **) new long double*[n];
   for (i=0;i<n;i++) b_ch[i]=(long double*) new long
double[n];
   y=(long double **) new long double*[n];
   for (i=0;i<n;i++) y[i]=(long double*) new long
double[n];
   value=h1[pos];
   for (i=0;i<n;i++)
      if (i==pos) h1[i]=1/h1[i];
         else h1[i]=-h1[i]/value;
   for (i=0;i<n;i++) {
      for (j=0;j<n;j++) {
         if (i!=pos) b_ch[i][j]=a1[i][j];
           else b_ch[i][j]=0;
      }
   }
   for (i=0;i<n;i++)
     for (j=0;j<n;j++)
        y[i][j]=h1[i]*a1[pos][j];
```

```
   for (i=0;i<n;i++)
     for (j=0;j<n;j++)
         a1[i][j]=y[i][j]+b_ch[i][j];

   for (i=0;i<n;i++) {
      delete b_ch[i];
      delete y[i];
   }
   delete y;
   delete b_ch;
}
```

ΚΕΦΑΛΑΙΟ 4
ΥΠΟΛΟΓΙΣΤΙΚΗ ΜΕΛΕΤΗ

Με την υλοποίηση ενός αλγορίθμου ο κατασκευαστής του οφείλει να κατασκευάσει μια υπολογιστική μελέτη στην οποία θα φαίνεται η ταχύτητα του αλγορίθμου. Σε αυτήν καταγράφονται οι CPU χρόνοι για κάποιες διαδικασίες και άλλοι χρήσιμοι αριθμοί για τη εξαγωγή συμπερασμάτων. Το σύστημα που χρησιμοποιήθηκε για τις μετρήσεις είναι Pentium 4 στα 2.4 GHz με μνήμη 768MB στα 266 MHz. Η καταγραφή των χρόνων εκτέλεσης εκτός από την αποτύπωση της απόδοσης του αλγορίθμου, βοηθάει και στην βελτιστοποίηση του κώδικα με τον εντοπισμό των κομματιών που καταλαμβάνουν τον περισσότερο χρόνο εκτέλεσης.

Στο κεφάλαιο αυτό παρατίθενται αναλυτικά στατιστικά, κάποια συμπεράσματα και προτάσεις για περαιτέρω έρευνα και βελτίωση της υλοποίησης του προγράμματος.

4.1 Στατιστικά εκτέλεσης προγραμμάτων

Το πρόγραμμα εκτελέστηκε σε δύο set δεδομένων, τα προβλήματα benchmark και κάποια τυχαία βέλτιστα προβλήματα δημιουργημένα από τον υπολογιστή. Η στατιστική μελέτη παρακολουθεί όλη την πορεία εκτέλεσης του προγράμματος από το διάβασμα των αρχείων mps μέχρι την επίλυση του αλγορίθμού. Βέβαια, τα στοιχεία που κατεγράφησαν δεν αφορούν και τις δύο ομάδες δεδομένων αφού στα τυχαία προβλήματα δεν εφαρμόστηκαν προλυτικές διαδικασίες.

Συνάρτηση mps to lp

NAME	READ				WRITE				TOTAL TIME
	ROWS	COLUMNS	RHS	RANGES	EQUIN	B	C	A	
optimal									
25fv47	0.000	0.040	0.010	0.000	0.000	0.000	0.030	0.070	0.150
adlittle	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.010	0.010
afiro	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
agg	0.000	0.040	0.010	0.000	0.000	0.000	0.010	0.010	0.070
agg2	0.000	0.060	0.020	0.000	0.000	0.000	0.010	0.030	0.120
agg3	0.000	0.050	0.020	0.000	0.000	0.000	0.010	0.030	0.120
bandm	0.000	0.010	0.000	0.000	0.000	0.000	0.010	0.020	0.040

beaconfd	0.000	0.020	0.000	0.000	0.000	0.000	0.010	0.020	0.050
blend	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
bnl1	0.000	0.050	0.000	0.000	0.000	0.010	0.010	0.040	0.110
bnl2	0.010	0.490	0.020	0.000	0.010	0.000	0.050	0.100	0.680
brandy	0.000	0.010	0.000	0.000	0.000	0.000	0.010	0.010	0.030
d2q06c	0.000	0.230	0.040	0.000	0.000	0.010	0.100	0.230	0.610
degen2	0.000	0.030	0.010	0.000	0.000	0.000	0.020	0.020	0.070
degen3	0.100	0.150	0.030	0.000	0.000	0.010	0.060	0.160	0.420
e226	0.000	0.010	0.000	0.000	0.000	0.000	0.010	0.020	0.040
fffff800	0.000	0.060	0.000	0.000	0.000	0.000	0.020	0.040	0.120
israel	0.000	0.000	0.000	0.000	0.000	0.000	0.010	0.010	0.020
lotfi	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
maros-r7	0.010	1.091	0.331	0.000	0.000	0.010	0.400	0.941	2.803
qap12	0.000	0.801	0.000	0.000	0.000	0.000	0.140	0.270	1.231
qap15	0.010	7.650	0.020	0.000	0.000	0.000	0.361	0.661	8.742
qap8	0.000	0.060	0.000	0.000	0.000	0.000	0.010	0.060	0.140
sc105	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
sc205	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
sc50a	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
sc50b	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
scagr7	0.000	0.000	0.000	0.000	0.000	0.000	0.010	0.000	0.010
scfxm1	0.000	0.010	0.000	0.000	0.000	0.000	0.010	0.020	0.040
scfxm2	0.000	0.030	0.010	0.000	0.000	0.000	0.010	0.030	0.080
scfxm3	0.000	0.060	0.010	0.000	0.000	0.010	0.010	0.040	0.130
scagr25	0.000	0.010	0.010	0.000	0.000	0.000	0.010	0.010	0.040
scorpion	0.000	0.020	0.000	0.000	0.000	0.000	0.010	0.010	0.040
scrs8	0.000	0.020	0.000	0.000	0.000	0.000	0.010	0.030	0.060
scsd1	0.000	0.000	0.000	0.000	0.000	0.000	0.010	0.020	0.030
scsd6	0.000	0.010	0.000	0.000	0.000	0.000	0.010	0.030	0.050
scsd8	0.000	0.050	0.000	0.000	0.000	0.000	0.040	0.060	0.160
sctap1	0.000	0.010	0.000	0.000	0.000	0.000	0.010	0.010	0.030
sctap2	0.000	0.060	0.020	0.000	0.000	0.010	0.020	0.050	0.160
sctap3	0.000	0.080	0.040	0.000	0.000	0.010	0.040	0.060	0.220
share1b	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
share2b	0.000	0.000	0.000	0.000	0.000	0.000	0.010	0.000	0.010
ship04l	0.010	0.060	0.010	0.000	0.000	0.000	0.030	0.050	0.160
ship04s	0.000	0.050	0.000	0.000	0.000	0.000	0.020	0.030	0.100
ship08l	0.000	0.180	0.020	0.000	0.000	0.010	0.060	0.100	0.370
ship08s	0.000	0.110	0.020	0.000	0.000	0.000	0.030	0.060	0.220
ship12l	0.000	0.320	0.040	0.000	0.000	0.000	0.080	0.130	0.570
ship12s	0.000	0.170	0.040	0.000	0.000	0.000	0.030	0.070	0.320
stocfor1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.010	0.010

stocfor2	0.010	0.130	0.000	0.000	0.010	0.000	0.030	0.060	0.240
stocfor3	0.020	14.550	0.020	0.000	0.010	0.010	0.231	0.470	15.351
truss	0.010	1.011	0.010	0.000	0.000	0.000	0.130	0.220	1.392
wood1p	0.000	0.210	0.000	0.000	0.000	0.000	0.170	0.411	0.811
woodw	0.000	0.460	0.000	0.000	0.000	0.000	0.110	0.261	0.841
Infeasible									
BGPRTR	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
ITEST2	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
ITEST6	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
KLEIN1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.010	0.010
KLEIN2	0.000	0.010	0.000	0.000	0.010	0.000	0.010	0.020	0.050
KLEIN3	0.000	0.020	0.000	0.000	0.000	0.000	0.030	0.070	0.120
kennington									
cre_a	0.000	0.981	0.030	0.000	0.000	0.000	0.070	0.110	1.201
cre_b	0.000	72.243	0.240	0.000	0.010	0.010	1.152	1.932	75.718
cre_c	0.010	0.520	0.020	0.000	0.010	0.000	0.060	0.091	0.721
cre_d	0.010	58.914	0.240	0.000	0.010	0.000	1.102	1.812	62.218
osa-07	0.000	2.423	0.090	0.000	0.000	0.000	0.521	0.971	4.055
osa-14	0.000	10.284	0.351	0.000	0.010	0.000	1.141	2.163	14.059

Δημιουργία τυχαίων προβλημάτων

TYPE	SIZE			EXECUTION TIME		
Optimal	Density	Rows	Columns	Generate Matrix	Writing File	Total Time
NAI	100%	10	10	0.010	0.039	0.049
NAI	100%	100	100	0.010	0.079	0.089
NAI	100%	500	500	0.220	1.600	1.820
NAI	100%	1000	1000	0.841	5.986	6.827
NAI	100%	3000	3000	8.812	54.497	63.309
'OXI	100%	10	10	0.000	0.009	0.009
'OXI	100%	100	100	0.010	0.069	0.079
'OXI	100%	500	500	0.200	1.440	1.640
'OXI	100%	1000	1000	0.771	5.866	6.637
'OXI	100%	3000	3000	8.221	53.795	62.016
NAI	5%	1000	1000	0.150	0.459	0.609
NAI	10%	1000	1000	0.190	0.729	0.919
NAI	20%	1000	1000	0.290	1.310	1.600
NAI	10%	3000	3000	1.922	6.587	8.509
NAI	20%	3000	3000	3.034	12.286	15.320

ΌΧΙ	5%	1000	1000	0.090	0.369	0.459
ΌΧΙ	10%	1000	1000	0.130	0.479	0.609
ΌΧΙ	20%	1000	1000	0.260	1.270	1.530
ΌΧΙ	10%	3000	3000	1.502	6.147	7.649
ΌΧΙ	20%	3000	3000	2.663	11.795	14.458

Προλυτικές διαδικασίες

NAME	Total Time	Scaling Time	Iterations	Size Before	Size After	Empty rows	Empty columns	Singleton Rows	Singleton columns	Implied bound	Redundant columns
optimal											
25fv47	21.06	2.03	3	822 x 1571	790 x 1542	0,0	0,0	26,25	4,4	1,0	0,0
Adlittle	0.08	0.00	2	57 x 97	56 x 95	0,0	0,0	1,1	0,1	0,0	0,0
Agg	3.94	0.66	2	489 x 163	469 x 163	0,0	0,0	1,0	0,0	19,0	0,0
agg2	7.42	1.10	2	517 x 302	515 x 301	0,0	0,0	1,1	0,0	1,0	0,0
agg3	7.48	1.16	2	517 x 302	515 x 301	0,0	0,0	1,1	0,0	1,0	0,0
Bandm	4.05	0.03	5	306 x 472	248 x 412	2,0	0,0	50,50	6,6	0,0	0,4
Beaconfd	2.97	0.00	4	174 x 262	105 x 193	0,0	0,4	27,27	38,38	4,0	0,0
Blend	0.10	0.00	2	75 x 83	72 x 80	0,0	0,0	0,0	3,3	0,0	0,0
bnl1	22.82	0.50	6	644 x 1175	619 x 1168	11,0	0,0	14,6	0,1	0,0	0,0
Brandy	1.41	0.00	3	221 x 249	135 x 207	37,0	0,0	44,44	1,1	3,0	0,0
d2q06c	286.01	31.41	3	2172 x 5167	2132 x 5124	0,0	0,3	10,10	30,30	0,0	0,0
e226	2.52	0.06	3	224 x 282	199 x 264	1,0	0,0	15,11	1,2	8,0	0,5
fffff800	16.03	0.13	3	525 x 854	323 x 663	0,0	0,0	37,37	154,154	11,0	0,0
Lotfi	1.05	0.02	5	154 x 308	134 x 288	0,0	0,0	8,8	12,12	0,0	0,0
sc105	0.07	0.00	2	106 x 103	105 x 103	1,0	0,0	0,0	0,0	0,0	0,0
sc205	0.25	0.00	2	206 x 203	204 x 202	1,0	0,0	1,1	0,0	0,0	0,0
sc50a	0.02	0.00	2	51 x 48	50 x 48	1,0	0,0	0,0	0,0	0,0	0,0
sc50b	0.01	0.01	2	51 x 48	49 x 48	2,0	0,0	0,0	0,0	0,0	0,0
scagr7	0.13	0.01	2	130 x 140	128 x 138	0,0	0,0	1,1	1,1	0,0	0,0
scfxm1	1.85	0.06	2	331 x 457	311 x 440	0,0	0,0	10,9	8,8	2,0	2,0
scfxm2	7.30	0.37	2	661 x 914	621 x 880	0,0	0,0	20,18	16,16	4,0	0,0
scfxm3	16.52	1.23	2	991 x 1371	931 x 1320	0,0	0,0	30,27	24,24	6,0	0,0
scagr25	1.52	0.03	2	472 x 500	470 x 498	0,0	0,0	1,1	1,1	0,0	0,0
scorpion	0.54	0.06	2	389 x 358	358 x 334	0,0	0,0	30,23	1,1	0,0	0,0
scrs8	18.63	0.05	17	491 x 1169	424 x 1107	0,0	0,0	33,32	31,31	4,0	0,0
sctap1	2.35	0.02	2	301 x 480	285 x 480	0,0	0,0	0,0	0,0	16,0	0,0
sctap2	36.88	0.12	2	1091 x 1880	1034 x 1880	0,0	0,0	0,0	0,0	57,0	0,0
sctap3	65.55	0.21	2	1481 x 2480	1409 x 2480	0,0	0,0	0,0	0,0	72,0	0,0
share1b	0.24	0.01	2	118 x 225	113 x 220	0,0	0,0	5,5	0,0	0,0	0,0
ship04l	2.32	0.11	2	403 x 2118	349 x 2114	40,0	0,0	4,4	0,0	8,0	0,0

ship04s	4.50	0.05	2	403 x 1458	261 x 1366	40,0	0,0	92,92	0,0	8,0	0,0
ship08l	9.33	0.38	2	779 x 4283	681 x 4259	64,0	0,0	24,24	0,0	8,0	0,0
ship08s	20.35	0.12	2	779 x 2387	409 x 2091	64,0	0,0	296,296	0,0	8,0	0,0
ship12l	38.46	0.67	2	1152 x 5427	834 x 5223	107,0	0,0	204,204	0,0	5,0	0,0
ship12s	43.49	0.14	2	1152 x 2763	461 x 2187	107,0	0,0	576,576	0,0	5,0	0,0
stocfor1	0.22	0.01	4	118 x 111	103 x 96	0,0	0,0	11,11	4,4	0,0	0,0
stocfor2	72.04	0.20	4	2158 x 2031	1981 x 1854	0,0	0,0	16,16	161,161	0,0	0,0
wood1p	12.25	0.53	2	245 x 2594	244 x 2593	0,0	0,0	0,0	1,1	0,0	0,0
woodw	255.06	1.01	3	1099 x 8405	900 x 6857	195,0	0,0	0,0	4,4	0,0	0,1544
infeasible											
BGPRTR	0.01	0.00	1	21 x 34	21 x 34	0,0	0,0	0,0	0,0	0,0	0,0
ITEST2	0.00	0.00	1	10 x 4	10 x 4	0,0	0,0	0,0	0,0	0,0	0,0
ITEST6	0.01	0.00	3	12 x 8	8 x 3	0,0	0,0	0,0	2,5	2,0	0,0
KLEIN1	0.07	0.01	1	55 x 54	55 x 54	0,0	0,0	0,0	0,0	0,0	0,0
KLEIN2	4.05	1.05	1	478 x 54	478 x 54	0,0	0,0	0,0	0,0	0,0	0,0
KLEIN3	22.28	12.78	1	995 x 88	995 x 88	0,0	0,0	0,0	0,0	0,0	0,0
kennington											
cre_a	235.83	4.82	3	3517 x 4067	3078 x 8029	88,0	0,0	251,7	0,0	0,0	0,0

Επίλυση τυχαίων βέλτιστών

Size	Density	Generating time	Read to mps	Eta matrix		Rank one matrix	
				Solution time	Iterations	Solution time	Iterations
500 x 500	**0.5%**	0.079	0.040	34.441	663	30.996	663
500 x 500	**1.0%**	0.089	0.060	117.281	1840	105.971	1840
500 x 500	**2.0%**	0.099	0.080	274.649	4153	246.795	4153
600 x 600	**0.5%**	0.119	0.060	75.959	927	67.263	927
600 x 600	**1.0%**	0.129	0.080	362.857	3855	324.639	3855
600 x 600	**2.0%**	0.159	0.120	719.455	7555	645.679	7555
700 x 700	**0.5%**	0.129	0.080	203.168	1675	179.145	1675
700 x 700	**1.0%**	0.139	0.110	832.476	6167	726.034	6167
700 x 700	**2.0%**	0.189	0.150	1435.893	11074	1286.530	11074
800 x 800	**0.5%**	0.179	0.090	432.787	2652	382.194	2652
800 x 800	**1.0%**	0.199	0.130	1404.950	8320	1293.420	8320
800 x 800	**2.0%**	0.239	0.200	2554.930	15216	2319.560	15216
900 x 900	**0.5%**	0.219	0.120	1010.420	4696	876.580	4696
900 x 900	**1.0%**	0.229	0.160	2541.870	12054	2276.470	12054
900 x 900	**2.0%**	0.299	0.250	2578.770	22418	2086.580	22418
1000 x 1000	**0.5%**	0.259	0.150	1663.910	6324	1493.220	6324
1000 x 1000	**1.0%**	0.259	0.190	2562.760	17553	2015.980	17553
1000 x 1000	**2.0%**	0.352	0.300	5446.520	28890	4688.320	28890

Επίλυση Benchmark

	lprref			eta-matrix inversion				Rank one inversion			
NAME	Time	Slack variables Added		Time	Iterations	Tolerance		Time	Iterations	Tolerance	
optimal											
adlittle	0.000	41		0.090	122	n*		0.060	123	n*	
afiro	0.000	19		0.000	12	n		0.000	12	n	
agg	0.110	432		5.378	105	n		4.817	105	n	
agg2	0.190	454		7.240	135	n		6.388	135	n	
agg3	0.190	454		8.271	153	n		7.230	153	n	
bandm	0.211	0		5.417	412	n		4.876	412	n	
beaconfd	0.050	27		0.070	33	n		0.040	33	n	
blend	0.020	31		0.050	70	n		0.050	70	n	
brandy	0.140	35		1.291	320	n		1.041	320	n	
e226	0.030	169		4.957	524	n*		4.416	524	n*	
fffff800	0.651	163		15.441	568	n*		13.989	568	n*	
israel	0.010	174		2.163	332	n		1.892	332	n	
lotfi	0.050	58		1.022	248	n		0.861	248	n	
sc105	0.020	59		0.130	76	n		0.140	76	n	
sc205	0.090	113		1.392	170	n		1.242	170	n	
sc50a	0.010	29		0.020	33	n		0.010	33	n	
sc50b	0.000	28		0.010	37	n		0.000	37	n	
scagr7	0.050	45		0.290	101	n		0.250	101	n	
scfxm1	0.311	139		8.261	364	n		7.029	364	n	
scfxm2	2.033	278		71.602	780	1.0e-8*		80.094	858	1.0e-8*	
scfxm3	6.309	417		337.000	1246	1.0e-8*		309.554	1246	1.0e-8*	
scrs8	1.202	96		25.666	548	1.0e-8*		23.233	548	1.0e-8*	
sctap1	0.120	164		7.120	362	1.0e-8*		6.479	362	1.0e-8*	
sctap2	4.677	563		252.673	902	1.0e-8*		218.253	902	1.0e-8*	
share1b	0.060	28		0.731	238	n		0.551	238	n	
share2b	0.010	83		0.170	103	n*		0.140	103	n*	
ship04l	1.322	40		17.935	437	n*		16.804	439	1.0e-8*	
ship04s	0.541	40		5.798	280	n*		5.347	280	n	
ship08l	8.172	72		105.411	679	n		97.030	679	n	
ship08s	1.592	72		19.017	369	n		17.365	369	n	
ship12l	14.400	101		414.185	1687	n*		374.267	1674	n*	
ship12s	2.053	101		80.632	747	n		46.546	747	n	
stocfor1	0.020	54		0.130	60	n		0.080	60	n	
stocfor2	25.568	1014		1088.640	1051	n		942.144	1051	n	

infeasible										
BGPRTR	0.000	6		0.000	14	n		0.000	14	n
ITEST2	0.000	9		0.000	4	n		0.000	4	n
ITEST6	0.000	5		0.000	4	n		0.000	4	n
KLEIN1	0.010	54		0.070	131	n		0.070	131	n
KLEIN2	0.040	477		13.159	286	n		11.396	286	n
KLEIN3	0.120	994		151.537	758	n		131.889	758	n

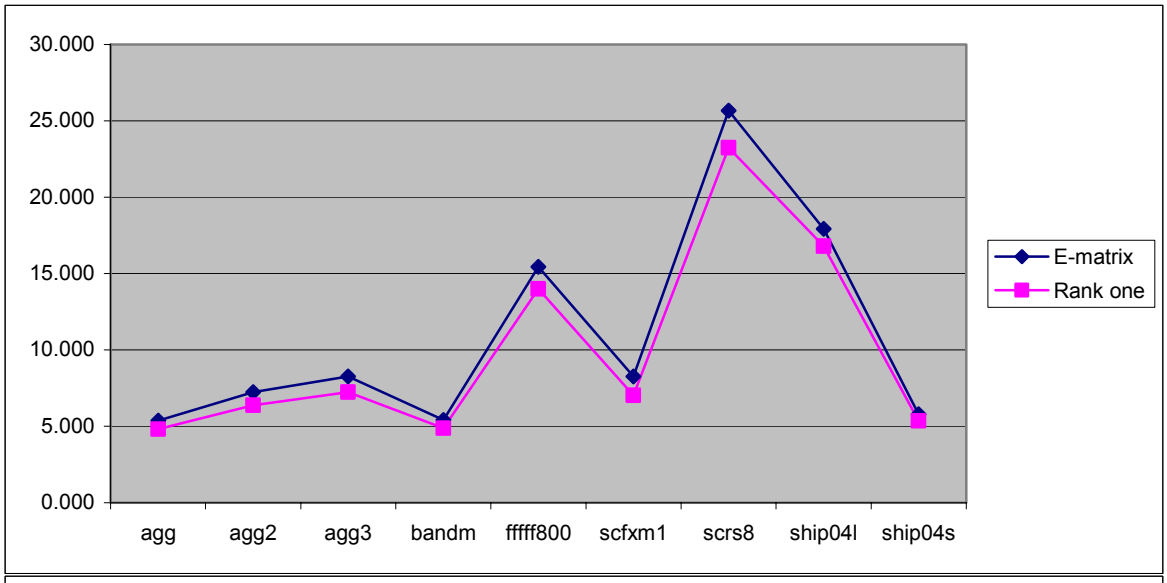
Speed up [*] ***Benchmark***

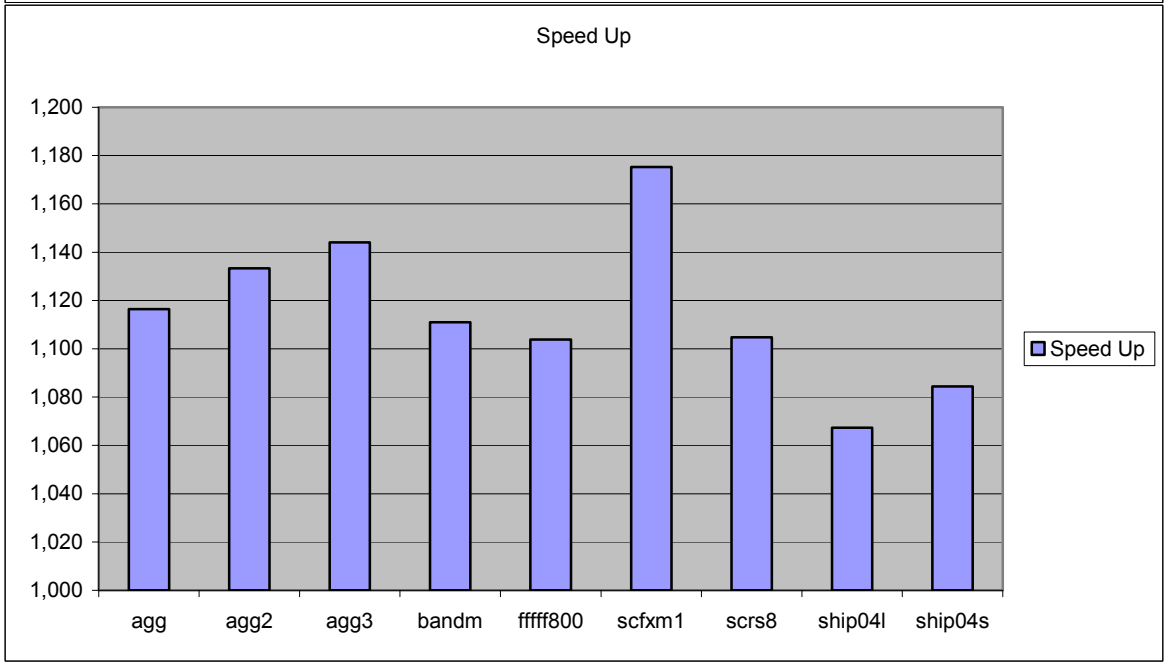
NAME	speedup
optimal	
adlittle	1,500
afiro	1,000
agg	1,116
agg2	1,133
agg3	1,144
bandm	1,111
beaconfd	1,750
blend	1,000
brandy	1,240
e226	1,123
fffff800	1,104
israel	1,143
lotfi	1,185
sc105	0,929
sc205	1,121
sc50a	2,000
sc50b	1,000
scagr7	1,160
scfxm1	1,175
scfxm2	0,894
scfxm3	1,089
scrs8	1,105
sctap1	1,099
sctap2	1,158
share1b	1,327
share2b	1,214
ship04l	1,067
ship04s	1,084
ship08l	1,086

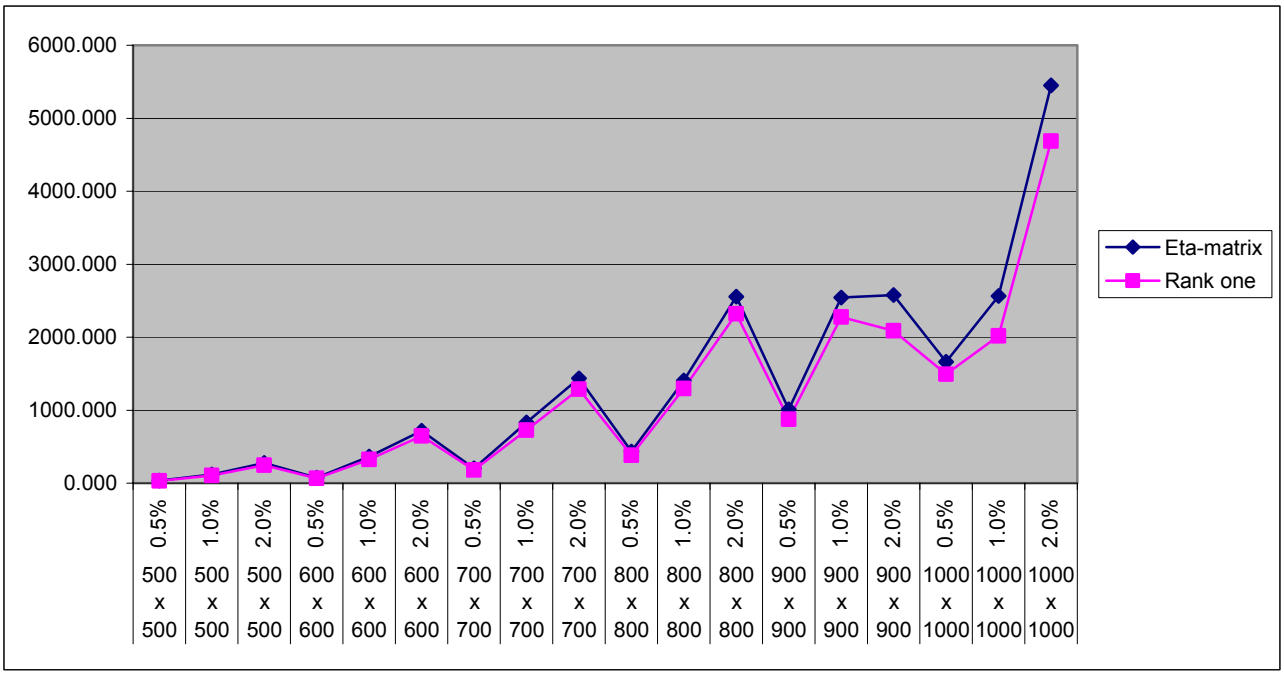
[*] speed up = CPUTime (eta-matrix) / CPUTime (rank one)

ship08s	1,095
ship12l	1,107
ship12s	1,732
stocfor1	1,625
stocfor2	1,155
infeasible	
BGPRTR	1,000
ITEST2	1,000
ITEST6	1,000
KLEIN1	1,000
KLEIN2	1,155
KLEIN3	1,149

Σύγκριση E-matrix inversion, Rank one inversion






6000.000
5000.000
4000.000
3000.000
2000.000
1000.000
0.000
0.5% 500 x 500
1.0% 500 x 500
2.0% 500 x 500
0.5% 600 x 600
1.0% 600 x 600
2.0% 600 x 600
0.5% 700 x 700
1.0% 700 x 700
2.0% 700 x 700
0.5% 800 x 800
1.0% 800 x 800
2.0% 800 x 800
0.5% 900 x 900
1.0% 900 x 900
2.0% 900 x 900
0.5% 1000 x 1000
1.0% 1000 x 1000
2.0% 1000 x 1000
Eta-matrix
Rank one

ΚΕΦΑΛΑΙΟ 5
ΣΥΜΠΕΡΑΣΜΑΤΑ

Γνωρίζοντας ότι για την ικανοποιητική απόδοση του αλγορίθμου θα πρέπει να χρησιμοποιηθεί κάποια μέθοδος αντιστροφής με υπολογισμό από την προηγούμενη αντίστροφη, η εργασία αυτή αποσκοπούσε κατά κύριο λόγο στη σύγκριση δύο μεθόδων, την αντιστροφή με Ε-μήτρες και την αντιστροφή Rank one.

Η μέθοδος Rank one όπως και αναμενόταν με βάση την πολυπλοκότητα των δύο μεθόδων αποδείχθηκε ταχύτερη. Σε όλα τα benchmark προβλήματα όπως και στα τυχαία η μέθοδος ήταν πιο γρήγορη κατά ποσοστό περίπου 16% στο σύνολο των benchmark και 13% στα τυχαία. Παρατηρήθηκε επίσης ότι η διαφορά ανάμεσα στην εκτέλεση της μίας ή της άλλης μεθόδου ήταν ανεξάρτητη του μεγέθους του προβλήματος.

Το γεγονός ότι το speed up παίρνει τιμές από 1 έως 2 φανερώνει ότι σε κανένα πρόβλημα οι μέθοδος E-matrix δεν λειτουργεί καλύτερα από την Rank one. Για την ακρίβεια οι δύο μέθοδοι επιλύουν στον ίδιο χρόνο πάρα πολύ μικρά προγράμματα, ενώ σε όλα τα υπόλοιπα η μέθοδος Rank one επιφέρει καλύτερα αποτελέσματα. Η μεγαλύτερη βελτίωση εμφανίζεται στο πρόβλημα sc50a όπου το speed-up είναι 2.00 δηλαδή ο αλγόριθμος με την αντιστροφή Rank one επιλύει το πρόβλημα στον μισό χρόνο (0.1 sec) σε σχέση με την χρήση E-matrix (0.2sec). Στα 40 benchmark προβλήματα τα οποία επιλύονται το μέσο speed up είναι 1.176.

Σε ότι αφορά τις προλυτικές διαδικασίες τα περισσότερα προγράμματα εμφάνισαν την μεγαλύτερη μείωση στην διαγραφή singleton γραμμών. Πιο σπάνια εμφανίζονται οι πλεονασματικές μεταβλητές, ενώ την μικρότερη μείωση σε αριθμό γραμμών και στηλών εκτέλεσε το presolve διαγραφής κενών στηλών. Γενικότερα η υλοποίηση του preproccesing δεν θεωρείται ικανοποιητική από πλευράς CPU χρόνου γεγονός που αποδεικνύεται από το ότι κάποια μεγάλα προβλήματα δεν ολοκλήρωσαν την εκτέλεση τους.

Οι διαδικασίες δημιουργίας τυχαίων βέλτιστων προβλημάτων και διαβάσματος mps αρχείων είναι ικανοποιητικές και δεν υποφέρουν από προβλήματα ανάλογα με τις διαστάσεις των μητρών, ούτε από σπατάλη μνήμης. Το γεγονός αυτό αποδεικνύεται από το ότι ακόμα και τα μεγάλα προβλήματα πέρασαν από τις διαδικασίες αυτές γρήγορα και με επιτυχία.

Σε ότι αφορά τον προγραμματισμό της εφαρμογής χρησιμοποιήθηκαν ιδιαίτερα οι τεχνολογίες αραιών μητρών οι οποίες υλοποιήθηκαν με την χρήση δομής συνδεδεμένης λίστας. Η αποθήκευση αυτή διατηρήθηκε μέχρι τον αλγόριθμο όπου προτιμήθηκε αποθήκευση σε πίνακα με διπλό δείκτη, *long double***, για πιο γρήγορη και εύκολη

προσπέλαση. Με τη δομή δεδομένων linkedlist δόθηκε η δυνατότητα το μέγεθος της λίστας να αυξάνει δυναμικά και να μην είναι απαραίτητη η συνεχής δέσμευση και αποδέσμευση χώρου στη μνήμη. Έτσι, μπόρεσαν να φορτωθούν προγράμματα κάθε μεγέθους χωρίς αλλαγή στον κώδικα ή πέρασμα παραμέτρων, με μειονέκτημα όμως την αργή προσπέλαση σε σχέση με την κλασική λίστα στην οποία όμως ο χώρος που καταλαμβάνει στη μνήμη ορίζεται κάποια στιγμή στο πρόγραμμα (είτε με την συνάρτηση malloc είτε με την συνάρτηση new).

5.1 Προτάσεις για περαιτέρω μελέτη - βελτίωση

Παρ' ότι οι αλγόριθμοι τύπου Simplex δεν έχουν δυνατότητες βελτίωσης από αλγοριθμικής άποψης, στη συγκεκριμένη εφαρμογή υπάρχουν δυνατότητες βελτίωσης στον τρόπο υλοποίησης.

Σαν πρώτη και σημαντικότερη βελτίωση προτείνεται η προσπάθεια παραλληλοποίησης του αλγορίθμου. Μια τέτοια προσπάθεια απαιτεί ανάλυση των βημάτων και παράλληλη εκτέλεση μέρους αυτών διότι ο αλγόριθμος δεν μπορεί να εκτελεστεί 100% παράλληλα λόγω της φύσης του. Όμως, θα μπορούσαν να υπάρξουν πολύ καλά αποτελέσματα με την παραλληλοποίηση τμημάτων όπως η αντιστροφή της μήτρας B, ο υπολογισμός εισερχόμενης και εξερχόμενης μεταβλητής ή ακόμα και των προλυτικών διαδικασιών.

Σε ότι αφορά την ακολουθιακή υλοποίηση θα μπορούσαν να χρησιμοποιηθούν τεχνικές ανανέωσης των διανυσμάτων x_B και s_N αντί για τον πλήρη υπολογισμό τους. Επιπλέον, θα μπορούσε να χρησιμοποιηθεί ένας διαφορετικός τρόπος αποθήκευσης στις προλυτικές διαδικασίες με μια κλάση matrix η οποία θα εμπεριείχε δύο αποθηκεύσεις CSC και CSR ώστε να χρησιμοποιείται κάθε φορά η καταλληλότερη, όπως και μια τεχνολογία αποθήκευσης αραιών μητρών στον αλγόριθμο γεγονός το οποίο θα επέτρεπε επίλυση μεγαλύτερων προβλημάτων καθώς και μείωση του CPU χρόνου που απαιτείται για την εκτέλεση των συναρτήσεων του αλγορίθμου και της διαδικασίας lprref.

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Vanderbei, Robert J (1997). *Linear Programming Foundations and extension*, Kluwer Academic Publishers

2. Swanson Leonard W. Swanson (1985). *Linear Programming basic theory and applications*, McGraw-Hill Kogakusha ltd

3. Bazaraa Mokhtar και Jarvis John (1977). *Linear Programming and network flows*, John Wiley & Sons Inc

4. Γιάννης Σίσκος (1998). *Γραμμικός Προγραμματισμός*, Εκδόσεις νέων τεχνολογιών

5. Saul Ed. (1986). *Linear Programming 5th edition,* I.Gass

6. Στεφανίδης Γεώργιος και Σαμαράς Νικόλαος (1999). *Υπολογιστικές μέθοδοι με το Matlab*, Ζυγός

7. Παπαρίζος Κωνσταντίνος (1999). *Γραμμικός Προγραμματισμός Αλγόριθμοι και εφαρμογές,* Ζυγός

8. Samaras, N. (2001). *Computational improvements and efficient implementation of two path pivoting algorithms*, PhD Dissertation, Dept. of Applied Informatics, University of Macedonia, Economic and Social Sciences. (In Greek)

9. Fourer Robert και David M. Gay (1994) "*Experience with a primal presolve algorithm*", http://www.ampl.com/REFS/pripre.pdf Bixby, E.R. (1992). "*Implementing the simplex method: The initial basis*", ORSA Journal on Computing, Vol. 4, pp. 267-284.

10. Bland, G.R. (1977). "*New finite pivoting rules for the simplex method*", Mathematics of Operations Research, Vol. 2, pp. 103-107.

11. Dantzig, B.G. (1948). "*Programming in a linear structure*", Comptroller, US Air Force, Washington, D.C.

12. Dantzig, B.G. (1963). "*Linear programming and extensions*", Princeton University Press, Princeton.

13. Forrest, J.J.H., Tomlin, J.A. (1990). "*Vector processing in simplex and interior methods for linear programming*", Annals of Operations Research, Vol. 22, pp. 71-100.

14. Gay, D.M. (1985). "*Electronic mail distribution of linear programming test problems*", Mathematical Programming Society COAL Newsletter, Vol. 13, pp. 10-12.

15. Goldfarb, D., Reid, K.J. (1977). "*A practicable steepest-edge simplex algorithm*", Mathematical Programming, Vol. 12, pp. 361-371.

16. McCall, E.H. (1982). "*Performance results of the simplex algorithm for a set of real-world programming models*", Communications of the ACM, Vol. 25(3), pp. 207-212.

17. Tomlin, A.J. (1975). "*On scaling linear programming problems*", Mathematical Programming Study, Vol. 4, pp. 146-166.

18. Tomlin, A.J., Welch, S.J. (1986). "*Finding duplicate rows in a linear program*", Operations Research Letters, Vol. 5, pp. 7-11.

19. Wagner, H.M. (1957). "*A comparison of the original and revised simplex methods*", Operations Research, Vol. 5(3).