ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΟΙΚΟΝΟΜΙΚΩΝ ΚΑΙ ΚΟΙΝΩΝΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΑΡΑΛΛΗΛΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΛΓΟΡΙΘΜΩΝ ΓΙΑ ΠΡΟΒΛΗΜΑΤΑ ΓΡΑΜΜΙΚΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

EL-SAID M. BADR

ΕΠΙΒΛΕΠΟΥΣΑ ΕΠΙΤΡΟΠΗ

1. **Κωνσταντίνος Παπαρριζος, Καθηγητής(Επιβλέπων)**
2. **Κωνσταντίνος Μαργαρίτης, Καθηγητής (Μέλος)**
3. **Σατρατζέμη Μαρία, Αναπληρώτρια Καθηγήτρια (Μέλος)**

Θεσσαλονίκη, 2006

Άρθρα Διατριβής

Badr, E., Paparrizos, K., Samaras, N., Sifaleras, A. (2005) . On the basis inverse of the exterior point simplex algorithm.17° Εθνικό Συνέδριο Ελληνικής Εταιρίας Επιχειρησιακών Ερευνών.

Badr, E., Paparrizos, K., Samaras, N., Baloukas, T. (2006)" Some computational results on the efficiency of an exterior point algorithm" 18° Εθνικό Συνέδριο Ελληνικής Εταιρίας Επιχειρησιακών Ερευνών.

Badr, E., Paparrizos, K., Samaras, N., Baloukas, T. (2005) " A web-based solver for the single source shortest path problem".18° Εθνικό Συνέδριο Ελληνικής Εταιρίας Επιχειρησιακών Ερευνών.

Badr, E., Paparrizos, K., Samaras, N. (2006) " Some computational results on MPI Parallel Implementation of Dense Simplex Method". 3rd International Computer Engineering Conference.Cairo, EGYPT (Submitted for publication).

Badr, E., Paparrizos, K., Samaras, N. (2006) "On Exterior Point Simplex Algorithm". European Journal of Operational Research (Submitted for publication).

Προγράμματα διατριβής

1- Πώς να υπολογίσετε τον αντίστροφο πίνακα
2- Πώς να μετατρέψετε την MPS μορφή σε LP μορφή
3- Μορφή πίνακα αλγόριθμου simplex
4- Επαναληπτικός Αλγόριθμος simplex
5- Εξωτερικός Αλγόριθμος simplex
6- Πώς να βρείτε το χρόνο της ΚΜΕ για τα προγράμματα αυτά
7- Πώς να αναπαραγάγετε τυχαία βέλτιστα LP προβλήματα (matlabe)
8- Μορφή πίνακα αλγόριθμου simplex ()
9- Επαναληπτικός Αλγόριθμος simplex από μονή τεχνητή μεταβλητή
10- Epsa eta_outer (matlabe)
11- Επαναληπτικός Αλγόριθμος simplex (matlabe)
12-Εξωτερικός Αλγόριθμος simplex (matlabe)
13-MPI πίνακας Αλγορίθμου simplex.

Αυτή η διατριβή αφιερώνεται στην οικογένειά μου, όλους αυτούς που μου έδωσαν κουράγιο και υποστήριξαν πριν και κατά διάρκεια αυτής της έρευνας.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1: ΕΙΣΑΓΩΓΗ 1

1.1. Εισαγωγή …… 1

1.2.Το Πρωτεύον Πρόβλημα……..….4

1.2.1 Ορολογία…….5

1.2.2 Ελαχιστοποίση και Μεγιστοποίηση……5

1.2.3 Χαλαρες and Surplus ματικές μεταβλητές ……6

1.3 ΤΟ Δυϊκό πρόβλημα……...7

1.4 Εφαρμογές Γραμμικού Προγραμματισμού ……10

1.4.1 Το Πρόβλημα Κατανομής των Πόρων……11

1.5 Γιατί ο Αλγόριθμος Simplex.....12

1.6 Γιατί ο Παράλληλος Υπολογισμός;.....13

1.6.1 Η ανάγκη του παράλληλου Υπολογισμού ……13

1.7 Οργάνωση της διατριβής…….….15

ΚΕΦΑΛΑΙΟ 2: ΑΛΓΟΡΙΘΜΟΙ ΤΥΠΟΥ SIMPLEX 17

2.1 Εισαγωγή ……17

2.2 Πώς δουλεύει γεωμετρικά η Μέθοδος Simplex..... 18

2.3 Πρωτεύον Αλγόριθμος Simplex ……...20

2.4 Δυϊκός Αλγόριθμος Simplex ……....29

2.5 Αλγόριθμος Simplex εξωτερικού Σημείου ……..37

2.6 Πρωτεύον Εξωτερικός Αλγόριθμος Simplex ……..38

2.7 Δυϊκός Αλγόριθμος Simplex ……46

ΚΕΦΑΛΑΙΟ 3:ΕΠΑΝΑΛΗΠΤΙΚΟΙ ΑΛΓΟΡΙΘΜΟΙ ΤΥΠΟΥ SIMPLEX 55
3.1 Εισαγωγή…….55
3.2 Άλγεβρα του Αλγόριθμου Simplex…….……56
3.3 Ο επαναληπτικός Αλγόριθμος Simplex …….……58
3.4 Μια περιγραφή του επαναληπτικού αλγόριθμου simplex …….59
3.5 Κανόνες πειστροφής……..….61
3.5.1 Κανόνας Dantizg……. 61
3.5.2 Κανόνας Bland ……62
3.5.3 Κανόνας Σωρού ……. 62
3.5.4 Κανόνας Ουράς…….. 62
3.5.5 Κανόνα του πιο πρόσφατο χρησιμοποιημένου στοιχείου……63
3.5.6 Κανόνας της μέγιστης ……63
3.5.7 Κανόνας Μερικής Εκτίμησης……..…63
3.5.8 Κανόνας της πιο Απότομης Ακμής ……..…64
3.6 Ανανέωση της αντιστροφής βάσεις…….....64
3.6.1 Ανάλυση των μεθόδων ανανέωσης……..…65
3.6.2. Επιστροφή μήτρα σε μορφή γινομένου……..…66
3.6.3 Μια τροποποίηση της μορφής γινομένου …….……..68
3.6.4 Ο Αλγόριθμου Simplex εξωτερικών σημείων…… ……....70
3.6.4.1 Περιγραφή Αλγορίθμου …… ……..…71
3.6.5 Υπολογιστική Μελέτη ……75
3.6.5.1 Συμπεράσματα……..77

ΚΕΦΑΛΑΙΟ 4: ΥΠΟΛΟΓΙΣΤΙΚΗ ΜΕΛΕΤΗ 78

4.1 Εισαγωγή……...……...78

4.2 Χαρακτηριστικά της Υπολογιστικής Μελέτης και Αρχιτεκτονικές ……...79
4.3 Αποτελέσματα της υπολογιστικής μελέτης……80

ΚΕΦΑΛΑΙΟ 5: ΠΑΡΑΛΛΗΛΟΣ ΥΠΟΛΟΓΙΣΜΟΣ 89

5.1 Εισαγωγή ……89
5.2 Ταξινόμηση Παράλληλων Υπολογιστών ……91
5.3 Έννοιες παράλληλου προγραμματισμού ……92
5.4 Εργαλεία Παράλληλου προγραμματισμού……..96
5.4.1 PVM……..97
5.4.2 MPI……...98
5.5 Απόδοση εργαλείων Αξιολόγησης ……...101
5.6 Διόρθωση ενός Παράλληλου προγράμματος……106
5.7 Βιβλιοθήκες παράλληλης βελτιστοποίησης ……..107
5.8 Απόδοση Μετρικές για Παράλληλα Συστήματα ……..109
5.9 Κίνητρο για σειριακή και κατανεμημένη υλοποίηση σε tableau format ……113
5.9.1 Γιατί η μέθοδος εφαρμόζεται σε tableau format……. ……113
5.9.1.1 Μη Υπολογισμός γραμμής κόστη και εισερχόμενη στήλης ……..113
5.9.1.2 Εναλλακτικοί κανόνες επιλογής στηλών μπορούν εύκολα να χρησιμοποιηθούν……...…….113
5.9.1.3 Η αναθεωρημένη μορφή δεν πλεονεκτεί στα πυκνά προβλήματα ……114
5.10 Εφαρμογές με πυκνές μήτρες…… ……114
5.11 Πρότυπα και Ανάλυση……115
5.11.1 Παράλληλη μέθοδος του Αλγόριθμος Simplex (PLPRG)……115
5.11.2 Συγχρονισμένες παράλληλες περιστροφές……. 118
5.12 Το πειραματικό περιβάλλον και αποτελέσματα ……119

5.12.1 Αρχιτεκτονική……...…119
5.12.2 Υπολογιστικά αποτελέσματα……120

Κεφάλαιο 6: ΣΥΖΗΤΗΣΗ ΚΑΙ ΠΡΟΤΑΣΕΙΣ 127

6.1 Εισαγωγή ……...127
6.2 Βελτιώσεις EPSA ……..127
6.3 Βελτιώσεις αντιστροφής της βάσης ……..127
6.4 Βελτιώσεις παράλληλου Αλγόριθμου Simplex ……...128

Παράρτημα ……...130

Αναφορές…….138

Ευχαριστίες

Επιθυμώ να εκφράσω το βαθύτερο συναίσθημα ευγνωμοσύνης και εκτίμησης στον επιβλέποντα καθηγητή της διδακτορικής διατριβής μου, τον κ. Κωνσταντίνο Παπαρρίζο για την ενθάρρυνση, τις χρήσιμες προτάσεις, τις εποικοδομητικές κριτικές και την καθοδήγησή του σε όλη τη μελέτη. Το συναίσθημα υποχρέωσής μου σ αυτόν είναι ένα συνεχές σημάδι της ακαδημαϊκής επιτυχίας του.

Θα ήθελα να εκφράσω τις πιο βαθιές ευχαριστίες και την εκτίμησή μου στα άλλα δύο μέλη της συμβουλευτικής επιτροπής, του καθηγητή Κωνσταντίνου Μαργαρίτη και της αναπληρώτριας καθηγήτριας Σατρατζέμη Μαρίας για τη συνεργασία και τη ζωτικής σημασίας υποστήριξη αυτής της διατριβής από την αρχή μέχρι την ολοκλήρωσή της.

Επιθυμώ να ευχαριστήσω το Δρ. Νικόλαο Σαμαρά επειδή έχω μάθει πολλά πράγματα για το γραμμικό προγραμματισμό από αυτό. Αυτή η διατριβή έχει ωφεληθεί από τις ιδέες, την ανατροφοδότηση, και την ενθάρρυνσή του.

Είμαι ευγνώμων στο ίδρυμα κρατικών υποτροφιών (Ι.Κ.Υ) της κυβέρνησης της Ελλάδας για την οικονομική ενίσχυσή κατά τη διάρκεια της μελέτης μου

Νιώθω επίσης υποχρεωμένος σε όλα τα μέλη και το προσωπικού του τμήματος Εφαρμοσμένης Πληροφορικής του πανεπιστημίου Μακεδονίας, Θεσσαλονίκη, Ελλάδα, για την υποστήριξή τους κατά τη διάρκεια της προετοιμασίας αυτής της διατριβής.

Τέλος, η επίτευξη εκπόνησης αυτής της διατριβής έγινε πολύ ευκολότερη μέσω της υποστήριξης και κατανόησης της οικογένειάς μου. Ήταν μια τεράστια πηγή δύναμης, ειδικά η μητέρα μου, η σύζυγός μου Gehan και τα παιδιά μου Sohail, Moaaz και Mosaab και ολόκληρη η οικογένεια που με τις ευχές τους και θυσίες τους με βοήθησαν να πετύχω.

El-Said M. Badr

Περίληψη

ΑΛΓΟΡΙΘΜΟΙ ΠΑΡΑΛΛΗΛΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΓΙΑ ΤΑ ΠΡΟΒΛΗΜΑΤΑ ΓΡΑΜΜΙΚΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

EL-SAID M. BADR

Επιβλέπων: **ΚΩΝΣΤΑΝΤΙΝΟΣ ΠΑΠΑΡΡΙΖΟΣ**

Υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων για την απόκτηση του Διδακτορικού Διπλώματος

(ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ)

2006

Η μέθοδος Simplex, η δημοφιλέστερη μέθοδος για τα γραμμικά προγράμματα (LPs), έχει δύο σημαντικές παραλλαγές. Είναι η αναθεωρημένη μορφή η μορφή του πλήρους tableau ή πλήρους πίνακα. Σήμερα, ουσιαστικά όλες οι σημαντικές υλοποιήσεις χρησιμοποιούν την αναθεωρημένη μορφή επειδή είναι περισσότερο αποτελεσματική σε αραιά LPs που είναι τα πιο κοινά.

Ωστόσο, η μέθοδος έχει επίσης πλεονεκτήματα. Κατ' αρχήν, ο πλήρης πίνακας μπορεί να είναι πολύ αποτελεσματικός για τα πυκνά προβλήματα. Δεύτερον, η μέθοδος του πλήρη πίνακα μπορεί εύκολα και αποτελεσματικά να επεκταθεί σε έναν κατανεμημένο αλγόριθμο. Ενώ τα πυκνά προβλήματα συναντιούνται σπάνια στην πράξη , εμφανίζονται συχνά σε μερικές σημαντικές εφαρμογές όπως στον ψηφιακό σχεδιασμό φίλτρων, την κατηγοριοποίηση κειμένων, την επεξεργασία εικόνας και την επίλυση προβλημάτων χρονοδρομολόγησης με τη μέθοδο των χαλαρώσεων των περιορισμών.

Υλοποιούμε δύο αλγορίθμους πλήρους πίνακα. Ο πρώτος, μια σειριακή εφαρμογή, είναι αποτελεσματικός για μικρά και μεσαίου μεγέθους πυκνά προβλήματα. Ο δεύτερος, μια απλή επέκταση του πρώτου, είναι ένας κατανεμημένος αλγόριθμος. Εφαρμόζουμε έναν αλγόριθμο που μετατρέπει την MPS μορφή σε μορφή LP και μετατρέπουμε ένα υποσύνολο των προβλημάτων συγκριτικής μέτρησης επιδόσεων από το NETLIB.

Παρουσιάζουμε μια ανάλυση δύο γνωστών σχημάτων ανάστροφης βάσης: (ι) Τη μορφή γινομένου της αντίστροφης (PFI) και (ιι) μια τροποποίηση της προηγούμενης μεθόδου (MPFI) και την ενσωματώνουμε με τον Αλγόριθμο Εξωτερικών Σημείων Τύπου Simplex (EPSA). Τα αποτελέσματα μιας υπολογιστικής μελέτης με ένα υποσύνολο των προβλημάτων συγκριτικής μέτρησης επιδόσεων από τη βιβλιοθήκη NETLIB δείχνουν ότι η μέθοδος (MPFI) είναι 1,60 φορές γρηγορότερα από τη μέθοδο (PFI) στα περισσότερα προβλήματα.

Εφαρμόζουμε τον αλγόριθμο simplex εξωτερικών σημείων. Αυτός ο αλγόριθμος φαίνεται να είναι αποδοτικότερος από τον κλασσικό πρωτεύοντα αλγόριθμο simplex (PSA), που υιοθετεί τον κανόνα Dantzig. Οι προκαταρκτικές υπολογιστικές μελέτες για τα τυχαία παραγόμενα αραιά γραμμικά προγράμματα υποστηρίζουν αυτήν την πεποίθηση. Αν και η υπολογιστική προσπάθεια που απαιτείται σε κάθε επανάληψη του EPSA απαιτεί περισσότερο χρόνο απ`ότι μια επανάληψη του PSA, η βελτίωση EPSA προέρχεται από το γεγονός ότι απαιτεί επαρκώς λιγότερες επαναλήψεις από τον PSA. Επιπλέον, καθώς το μέγεθος του προβλήματος αυξάνεται και οι μειώσεις πυκνότητας προβλήματος, ελαττώνεται ο EPSA γίνεται σχετικά γρηγορότερα.

Abstract

PARALLEL PROGRAMMING ALGORITHMS FOR LINEAR PROGRAMMING PROBLEMS

EL-SAID M. BADR

Advisor: **KONSTANTINIOS PAPRRIZOS**

Submitted in Partial Fulfillment of the Requirements for the degree of

DOCTOR OF PHILOSOPHY

(DEPARTMENT APPLIED INFORMATICS)

2006

The Simplex Method, the most popular method for solving Linear Programs (LPs), has two major variants. They are the revised method and the standard, or full tableau method. Today, virtually all serious implementations are of the revised method because it is more efficient for sparse LPs which are the most common.

However, the full tableau method has advantages as well. First, the full tableau can be very effective for dense problems. Second, a full tableau method can easily and effectively be extended to a coarse grained distributed algorithm. While dense problems are uncommon in general, they occur frequently in some important applications such as digital filter design, text categorization, image processing and relaxations of scheduling problems.

We implement two full tableau algorithms. The first, a serial implementation, is effective for small to moderately sized dense problems. The second, a simple extension of the

first, is a distributed algorithm. We implement an algorithm which converts MPS format to LP format and we convert a subset of benchmark problems from NETLIB.

We present an analysis of two well-known updating schemes for basis inverse: (i) The Product Form of the Inverse (PFI) and (ii) A Modification of the Product Form Inverse (MPFI) and incorporate it with EPSA. Computation results with a subset of benchmark problems from NETLIB indicate that the Modification of the Product Form of the Inverse is 1.60 times faster than the product form of the inverse in most problems.

We implement the exterior point simplex. This algorithm seems to be more efficient than the classical primal simplex algorithm (PSA), employing Dantzig's rule. Preliminary computational studies on randomly generated sparse linear programs support this belief. Although the computational effort required in each step of EPSA requires more time compared to an iteration step of PSA, the improvement of EPSA comes from the fact that it requires adequately less iterations than PSA. Moreover, as the problem size increases and the problem density decreases, EPSA gets relatively faster.

ΚΕΦΑΛΑΙΟ 1

ΕΙΣΑΓΩΓΗ

1.1 Εισαγωγή

Ο γραμμικός προγραμματισμός είναι ενας σχετικά νέος μαθηματικός κλάδος, που χρονολογείται από την εφεύρεση της μεθόδου Simplex από Γ. Β. Dantzig το 1947. Ιστορικά, η ανάπτυξη του γραμμικού προγραμματισμού επηρεάζεται κυρίως από τις εφαρμογές του στην οικονομία και τη διοίκηση. Ο Dantzig ανέπτυξε αρχικά τη μέθοδο Simplex για να λύσει τα προβλήματα προγραμματισμού της Πολεμικής Αεροπορίας των Η.Π.Α. και ο προγραμματισμός και ο σχεδιασμός κυριαρχούν ακόμα στις εφαρμογές του γραμμικού προγραμματισμού. Ένας λόγος για το οποίο ο γραμμικός προγραμματισμός είναι ένας σχετικά νέος τομέας είναι ότι μόνο τα πολύ μικρά προβλήματα γραμμικού προγραμματισμού μπορούν να λυθούν χωρίς υπολογιστή.

Η μέθοδος simplex παραμένει ακόμα μια από τις αποτελεσματικότερες μεθόδους για μια μεγάλη πλειοψηφία πρακτικών προβλημάτων. Αν και διάφορες παραλλαγές της μεθόδου simplex έχουν αναπτυχθεί, καμία από αυτές δεν έχει πολυωνυμική χρονική πολυπλοκότητα (Terlaky και Zhang 1993). Δηλαδή η μέθοδος simplex μπορεί να απαιτήσει υπολογιστική προσπάθεια που αυξάνεται εκθετικά με το μέγεθος των δεδομένων του προβλήματος. Ως εκ τούτου, η μέθοδος simplex δεν θεωρείται αποτελεσματική από θεωρητική άποψη αντίθετα προς την πρακτική αποτελεσματικότιτά της.

Ο πρώτος αποτελεσματικός αλγόριθμος του οποίου η υπολογιστική πολυπλοκότητα αυξάνεται πολυωνυμικά στο μέγεθος προβλήματος εφευρέθηκε από Khachian το 1979 και καλείται ελλειψοειδής αλγόριθμος. Αλλά η απόδοση αυτού του αλγορίθμου είναι κακή για τα πρακτικά προβλήματα συγκρινόμενα με τη μεθόδο simplex.

Το 1984, ο Karmarkar παρουσίασε έναν νέο πολυωνυμικό αλγόριθμο για LP που πλησιάζει τη βέλτιστη λύση από το εσωτερικό της εφικτής περιοχής (Karmarkar 1984). Από τότε που το κλασσικό άρθρο του Karmarkar δημοσιεύθηκε, πολλά άρθρα έχουν γραφτεί για τις μεθόδους εσωτερικού σημείου (Roos, 1997 και Wright, 1997). Όπως δείχνουν οι θεωρητικές ιδιότητές τους, οι μέθοδοι εσωτερικού σημείου λειτουργούν καλά για τα πρακτικά προβλήματα στην πραγματική ζωή και ξεπερνούν τη μέθοδο simplex για τα πολύ μεγάλης κλίμακας προβλήματα (Lustig και λοιποί 1994).

Παρά την κακή απόδοσή της για τα προβλήματα μεγάλης κλίμακας, η μέθοδος simplex λειτουργεί συχνά καλύτερα από τις μεθόδους εσωτερικού σημείου για τα μεσαία ή μικρού μεγεθούς προβλήματα. Επίσης, για μερικές κατηγορίες LP προβλημάτων όπως τα προβλήματα δικτυακής βελτιστοποίησης , η μέθοδος simplex απαιτεί πολύ λιγότερη υπολογιστική προσπάθεια από τις μεθόδους εσωτερικού σημείου. Επιπλέον, για τη μέθοδο λύσης μιας σειράς σχετικών LP προβλημάτων, η μέθοδος simplex είναι καταλληλότερη από τις μεθόδους εσωτερικού σημείου επειδή ο υπολογισμός ενός "θερμού" σημειού ξεκινήματος είναι δύσκολος για τις μεθόδους εσωτερικού σημείου ενώ η μέθοδος simplex δεν έχει καμία δυσκολία στον υπολογισμό του.

Ο αλγόριθμος εξωτερικού σημείου τύπου simplex (EPSA) αναπτύχθηκε αρχικά από Παπαρρίζο το 1991 για το πρόβλημα αντιστοίχισης. Αργότερα, το 1993, ο Παπαρρίζος γενίκευσε την μέθοδο εξωτερικού σημείου στο γενικό γραμμικό πρόβλημα με την ανάπτυξη ενός δυϊκού στη φύση αλγόριθμου. Στον πίνακα 1.1.1, ο Σαμαράς συνόψισε την ανάπτυξη στους αλγορίθμους τύπου Simplex (Samaras 2001).

No.	**Θεωρητική Ανάλυση**	**Συγγραφέας**	**Έτος**
1	Πρωτεύων Αλγόριθμος Simplex	G. B. Dantzig	1947
2	Δυαδική Θεωρία και δυαδικός Αλγόριθμος Simplex	C. E. Lemke	1954
3	a. Degeneracy and Cycling in L.P.	E. M. L. Beale K.T. Marshall, J. W. Suurballe	1955 1969

	b. Lexicographic Rules for Anticycling	G.B.Dantzig, A.Orden, P. Wolfe P. Wolfe	1955 1963
	c. Bland's Anticycling Rule	R. G. Bland	1977
4	Complexity of Simplex Algorithm and Worst-Case behaviour Probabilistic / Average behaviour	V.Klee, G.J. Minty R. G. Jeroslow K. H. Borgwardt S. Smale	1972 1973 1982 1983
5	Primal exterior point simplex	K. Paparrizos	1991

Πίνακας 1.1.1: Η ανάπτυξη Αλγοτίθμων τύπου Simplex

Στον πίνακα 1.1.2, ο Σαμαράς συνόψισε την ανάπτυξη στις μεθόδους εσωτερικού σημείου (Samaras 2001).

No.	**Θεωρητική Ανάλυση**	**Συγγραφέας**	**Έτος**
1	(Logarithmic barrier algorithm) Method of centers Affine scaling algorithm	K. Frisch P. Huard I. Dikin	1955 1967 1974
2	Karmarkar's algorithm	N. Karmarkar	1984
3	Recovering optimal solution	Y. Ye, M. Kojima	1987
4	Path following algorithm	M. Kojima, S. Mizuno, A. Yoshise	1989
5	Infeasible algorithms	I. Lustig	1990
6	Potential reduction algorithm	Y. Ye	1991
7	Positive semi definite programming	Y. Nesterov A. Nemirovskii	1993

Πίνακας 1.1.2: Η ανάπτυξη στις μεθόδους εσωτερικού σημείου.

Η αναθεωρημένη μέθοδος simplex είναι το όνομα μιας εφαρμογής της μεθόδου simplex που χρησιμοποιεί μια ελαφρώς διαφορετική μέθοδο ανανέωσης των δεδομένων σε κάθε επανάληψη: αντί της ανανέωσης του προβλήματος και τη χρήση των αποτελέσματων της προηγούμενης επανάληψης, χρησιμοποιεί τα αρχικά δεδομένα. Η αναθεωρημένη

μέθοδος simplex είναι πιό αποδοτική υπολογιστικά, ειδικά για τα μεγάλα και αραιά προβλήματα. Θα δούμε την περιγραφή της αναθεωρημένης μεθόδου simplex αργότερα.

Στον πίνακα 1.1.3 ο Σαμαράς εξήγησε τις κύριες βελτιώσεις υπολογισμού που σχετίζονται με τους αλγόριθμους τύπου simplex (Samaras 2001).

No.	Βελτιώσεις υπολογισμών	Συγγραφέας	Έτος
1	(Tableau format)	G. B. Dantzig	1947
2	(Revised format)	G. B. Dantzig, W. Orchard-Hays, P. Wolfe	1953-54
3	(Elimination form of the inverse)	M.H. Markowitz L.M.E. Beale E. Hellerman D. Rarick	1954 1971 1971 1972
4	(Sparse matrix updating techniques)	H. Bartels, H. Golub H. Forrest, A. Tomlin K. Reid	1969 1972 1976
5	(Pivoting techniques)	J. Harris D. Goldfarb, K. Reid H. Forrest, D. Goldfarb	1973 1977 1992
6	(Presolve procedures)	L. Brearley, Mitra, P. Williams	1975

Πίνακας 1.1.3: Οι κύριες υπολογιστικές βελτιώσεις των αλγορίθμων της μορφής simplex.

1.2 Το Πρωτεύον Πρόβλημα

Μπορούμε να γράψουμε τα γραμμικά προβλήματα ως εξής:

$$\begin{aligned} \max \quad & z = c^T x \\ \text{s.t.} \quad & Ax \otimes b \\ & x \geq 0 \end{aligned}$$

Όπου c, $x \in \Re^n$, $b \in \Re^m$, $A \in \Re^{mxn}$, c^T είναι το ανάστροφο ενός διανύσματος c και $\otimes$ αναφέρεται σε οποιαδήποτε από τις σχέσεις $(=, \leq \text{ and } \geq)$.

Επίσης μπορούμε να γράψουμε τα γραμμικά προγράμματα με έναν άλλο τρόπο:

$$\max \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to : } \sum_{j=1}^{n} a_{ij} x_j \otimes b_i \ , \qquad i=1 \ldots m$$

$$x_j \geq 0, \qquad j=1 \ldots n$$

1.2.1 Ορολογία

Η γραμμική συνάρτηση $c^T x$, την οποία προσπαθούμε να βελτιστοποιήσουμε, καλείται **αντικειμενική συνάρτηση**. Μια **εφικτή λύση** είναι μια ανάθεση τιμών στις μεταβλητές **απόφασης** $x_1, x_2, ..., x_n$, έτσι ώστε να ικανοποιούνται όλοι οι περιορισμοί. Μια εφικτή λύση που βελτιστοποιεί την αντικειμενική συνάρτηση είναι μια **βέλτιστη λύση.** Ένα γραμμικό πρόγραμμα που δεν έχει εφικτές λύσεις καλείται **αδύνατο,** έχει εφικτές λύσεις αλλά δεν έχει βέλτιστη λύση.

1.2.2 Ελαχιστοποίηση και Μεγιστοποίηση

Ένας άλλος χειρισμός προβλήματος είναι να μετατραπεί ένα πρόβλημα μεγιστοποίησης σε ένα πρόβλημα ελαχιστοποίησης και αντιθέτως. Σημειώστε ότι σε οποιαδήποτε περιοχή ισχύει η σχέση

$$\text{Μεγιστοποίηση} \sum_{j=1}^{n} c_j x_j = - \text{ ελαχιστοποίηση} \sum_{j=1}^{n} - c_j x_j$$

Έτσι ένα πρόβλημα μεγιστοποίησης (ελαχιστοποίησης) μπορεί να μετατραπεί σε ένα πρόβλημα ελαχιστοποίησης (μεγιστοποίησης) πολλαπλασιάζοντας τους συντελεστές της αντικειμενικής συνάρτησης με -1. Μετά τη βελτιστοποίηση του νέου προβλήματος, η βέλτιστη αντικειμενική τιμή του παλιού προβλήματος είναι -1 φορές της βέλτιστη τημή του νέου προβλήματος

1.2.3 Χαλαρές (Slack) και πλεονασματικές (Surplus) μεταβλητές

`Εστω, μια ανισότητα, για παράδειγμα του περιορισμού r, της ακόλουθης μορφής:

$$\sum_{j=1}^{n} a_{rj}x_j \le b_r \qquad (1.1)$$

Εισάγουμε μια νέα μεταβλητή , $s_r \ge 0$, που την ονομάζουμε χαλαρή μεταβλητή (slack), έτσι ώστε

$$\sum_{j=1}^{n} a_{rj}x_j + s_r = b_r \qquad (1.2)$$

Έτσι,

$$s_r = b_r - \sum_{j=1}^{n} a_{rj}x_j \qquad (1.3)$$

Περιφραστικά, s_r είναι η (μη αρνητική) διαφορά μεταξύ της σταθεράς του δεξιού μέλους και τιμής του αριστερού μέλους. Από μια μαθηματική άποψη, το s_r μας επιτρέπει να εκφράσουμε μια ανισότητα της μορφής (1.1) με ένα καταλληλότερο σχήμα ισότητας.

Στη συνέχεια, θεωρούμε μια ανισότητα για παράδειγμα περιορισμού t, της μορφής:

$$\sum_{j=1}^{n} a_{tj}x_j \ge b_t \qquad (1.4)$$

Στην περίπτωση αυτή, εισάγουμε μια νέα μεταβλητή, $s_t \ge 0$, έτσι ώστε

$$\sum_{j=1}^{n} a_{tj}x_j - s_t = b_t \qquad (1.5)$$

Σημειώνουμε ότι στην περίπτωση αυτή

$$s_t = \sum_{j=1}^{n} a_{rj}x_j - b_t \qquad (1.6)$$

Δηλαδή η μεταβλητή s_t αντιπροσωπεύει τη μη αρνητική διαφορά μεταξύ της αριστερής πλευράς και δεξιάς πλευράς της ανισότητας στην (1.4). Μια τέτοια μεταβλητή καλείται μεταβλητή πλεονάσματος (surplus) ή πλεονασματική μεταβλητή.

1.3 Το Δυϊκό πρόβλημα

Κάθε πρόβλημα μεγιστοποίησης γραμμικού προγραμματισμού έχει ένα ισοδύναμο πρόβλημα ελαχιστοποίησης, το δυϊκό πρόβλημα. Αυτά τα δύο προβλήματα παρέχουν χρήσιμες πληροφορίες το ένα για το άλλο. Παραδείγματος χάριν, η τιμή της δυϊκής αντικειμενικής συνάρτησης παρέχει έναν ανώτερο όριο για την τιμή της πρωταρχικής αντικειμενικής συνάρτησης. Αυτό και άλλα αποτελέσματα συζητούνται κατωτέρω.

Η κανονική μορφή ενός προβλήματος γραμμικού προγραμματισμού είναι μια μορφή στην οποία η αντικειμενική συνάρτηση μεγιστοποιείται, όλοι οι περιορισμοί είναι της μορφής($\geq$), και όλες οι μεταβλητές είναι μη αρνητικές. Συνεπώς, το πρωτεύον πρόβλημα, με κανονική μορφή είναι:

$$\begin{aligned} \max \quad & z = c^T x \\ \text{s.t.} \quad & Ax \geq b \\ & x \geq 0 \end{aligned} \qquad \text{(P)}$$

Εάν (P) διευκρινίζεται να είναι το πρωτεύον πρόβλημα γραμμικού προγραμματισμού, το δυϊκό πρόβλημα γραμμικού προγραμματισμού του δίνεται από (DP):

$$\begin{aligned} \min \quad & z = b^T w \\ \text{s.t.} \quad & Aw \leq c \\ & w \geq 0 \end{aligned} \qquad \text{(DP)}$$

Αναφέρουμε τώρα μερικές κανόνες υπολογισμού του δυϊκού προβλήματος.

1- Η αντικειμενική συνάρτηση του πρωτεύοντος μεγιστοποιείται, η αντικειμενική συνάρτηση δυϊκού ελαχιστοποιείται.

2- Το πρόβλημα μεγιστοποίησης πρέπει να έχει όλους τους περιορισμούς ($\geq$) και το πρόβλημα ελαχιστοποίησης πρέπει να έχει όλους τους περιορισμούς ($\leq$).

3- Όλες οι πρωτεύουσες και δυϊκές μεταβλητές πρέπει να είναι μη αρνητικές.

4- Κάθε περιορισμός σε ένα πρόβλημα αντιστοιχεί σε μια μεταβλητή του άλλου (και αντίστροφα).

5- Τα στοιχεία της δεξιάς πλευράς των περιορισμών σε ένα πρόβλημα είναι οι αντίστοιχοι συντελεστές της αντικειμενικής συνάρτησης στο άλλο πρόβλημα.

6-Ο πίνακας των σταθερών συντελεστών για ένα πρόβλημα είναι ο ανάστροφος του πίνακα σταθερών συντελεστών για το άλλο πρόβλημα

Χρησιμοποιώντας τις σχέσεις στον πίνακα 1.3.1, είναι δυνατό να γραφτεί το δυϊκό πρόβλημα για ένα δεδομένο γραμμικό πρόγραμμα χωρίς το ενδιάμεσο βήμα του μετασχηματισμού του προβλήματος στην κανονική μορφή.

Μεγιστοποίηση προβλήματος		**Ελαχιστοποίηση προβλήματος**
Σταθερές	$\leftrightarrow$	μεταβλητές
$\leq$	$\leftrightarrow$	≥ 0
$\geq$	$\leftrightarrow$	≤ 0
$=$	$\leftrightarrow$	Χωρίς περιορισμό
Μεταβλητές		σταθερές
≥ 0	$\leftrightarrow$	$\geq$
≤ 0	$\leftrightarrow$	$\leq$
Χωρίς περιορισμό	$\leftrightarrow$	$=$

Πίνακας 1.3.1: Πρωταρχικές-δυαδικές σχέσεις

Παράδειγμα 1.3.1

Να υπολογιστεί το δυϊκό πρόβλημα του επόμενου προβλήματος.

$$
\begin{array}{lrcrcrcr}
\text{Max} & 4x_1 & + & 2x_2 & - & x_3 & & \\
\text{subject to} & x_1 & + & x_2 & + & x_3 & = & 20 \\
 & 2x_1 & - & x_2 & & & \geq & 6 \\
 & 3x_1 & + & 2x_2 & + & x_3 & \leq & 40 \\
 & & & x_1 & & x_2 & \geq & 0 \\
 & & & x3 & & \text{unrestricted} & &
\end{array}
$$

Θέτοντας w_1, w_2 and w_3 τις δυϊκες μεταβλητές, τα αποτελέσματα του Πίνακα 1.3.1 παράγουν το ακόλουθο δυαδικό πρόβλημα:

$$
\begin{array}{lrcrcrcr}
\text{ελάχιστο} & 20w_1 & + & 6w_2 & + & 40w_3 & & \\
\text{subject to} & w_1 & + & 2w_2 & + & 3w_3 & \geq & 4 \\
 & w_1 & - & w_2 & + & w_3 & \geq & 2 \\
 & w_1 & + & & & w_3 & = & -1 \\
 & & & w_1 & & \text{unrestricted} & & \\
 & & & w_2 & & \leq 0 & & \\
 & & & w_3 & & \geq 0 & &
\end{array}
$$

Ο πίνακας 1.3.2 παρουσιάζει διαφορετικούς συνδυασμούς πιθανούς για ένα πρωτεύον - δυϊκό ζευγάρι. Το δυϊκό LP αντιπροσωπεύει έναν γραμμικό συνδυασμό των πρωτευόντων περιορισμών LP. Η σημασία της δυϊοκότητας, γνωστή ως Θεώρημα Δυϊοκότητας, είναι ότι κάθε εφικτή λύση του δυϊκού LP παράγει ένα όριο στη βέλτιστη τιμή του πρωτεύοντος LP. Το Θεώρημα Δυϊκότητας παρέχει σε μας τη δυνατότητα να βεβαιώσουμε εάν μια λύση σε ένα LP είναι βέλτιστη ή όχι.

		Δυαδικό LP		
		Βέλτιστο	*Απραγματοποίητο*	*Απεριόριστο*
Πρωτεύον LP	*Βέλτιστο*	Ναι	Όχι	Όχι
	Απραγματοποίητο	Όχι	Ναι	Ναι
	Απεριόριστο	Όχι	Ναι	Όχι

Πίνακας 1.3.2: Πρωτεύουσες - Δυϊκές δυνατότητες

Τα ακόλουθα θεωρήματα περιγράφουν τις σχέσεις μεταξύ των πρωτευόντων και δυϊκων προβλημάτων. Για μια πληρέστερη συζήτηση αυτών των θεωρημάτων, αναφερθείτε στο (Chvatal 1983).

Θεώρημα 1.3.1. (Ασθενής δυϊκότητα στον Γραμμικό Προγραμματισμό).

Κάθε τιμή της πρωτεύουσας αντικειμενικής συνάρτησης παρέχει ένα κάτω όριο για κάθε τιμή της δυϊκής αντικειμενικής συνάρτησης.

Θεώρημα 1.3.2. (Ισχυρή δυϊκότητα στο Γραμμικό Πρόγραμμα τις μό).

Εάν ένα πρωτεύον πρόβλημα έχει μια βέλτιστη λύση, το δυϊκο του έχει επίσης μια βέλτιστη λύση και οι βέλτιστες τιμές των δύο προβλημάτων συμπίπτουν

Θεώρημα 1.3.3. (Συμπληρωματική χαλαρότητα στο γραμμικό προγραμματισμό).

Στη βελτιστοποίηση ή ο περιορισμός του πρωτεύοντος γραμμικού προβλήματος είναι ενεργός ή η αντίστοιχη δυϊκη μεταβλητή είναι 0 ή και τα δύο. Επιπλέον, ή ο δυϊκος γραμμικός περιορισμός είναι ενεργός ή η αντίστοιχη πρωτεύουσα μεταβλητή είναι 0 ή και τα δύο.

Ορισμός 1.3.1. (Χάσμα δυϊκότητας).

Η διαφορά μεταξύ των πρωτευόντων και δυϊκών αντικειμενικών τιμών των πρωτευόντων και δυϊκών εφικτών λύσεων καλείται χάσμα δυαδικότητας (duality gap)

1.4 Εφαρμογές Γραμμικού Προγραμματισμού

Ο γραμμικός προγραμματισμός είναι ένα από τα πιό χρήσιμα και θεμελιώδη μαθηματικά πρότυπα προγραμματισμού που χρησιμοποιείται ευρέως στην επιχειρησιακή έρευνα και σε πολλούς άλλους τομείς των επιστήμων (Murty 1983). Η χρησιμότητα των γραμμικών και μαθηματικών προτύπων προγραμματισμού έχει καταδειχθεί κατά τη διάρκεια των τελευταίων πενήντα ετών. Ο προγραμματισμός μεταφορών, η επιλογή

χαρτοφυλακίων και η ανάλυση παραγωγικότητας είναι καλά παραδείγματα των εφαρμογών του μαθηματικού προγραμματισμού στην επιχειρησιακή έρευνα (Park 1999). Στην πραγματική ζωή, πολλές εφαρμογές των μαθηματικών προτύπων προγραμματισμού προκαλούν τα πολύ μεγάλης κλίμακας προβλήματα LP. Επομένως, είναι πολύ ουσιαστικό να υπάρξουν οι μέθοδοι και οι εφαρμογές τους που λύνουν τα μεγάλα προβλήματα LP αποτελεσματικά. Υπάρχουν πολλές εφαρμογές για το γραμμικό προγραμματισμό όπως:

(i) Στρατιωτικές Εφαρμογές

(ii) Βιομηχανικές Εφαρμογές

(iii) Εφαρμογές Η/Υ και

(iv) Οικονομικές Εφαρμογές.

Οι εφαρμογές γραμμικού προγραμματισμού αντιπροσωπεύουν συχνά προβλήματα στα οικονομικά, παρά στις επιστήμες ή την εφαρμοσμένη μηχανική. Ακολουθούν μερικές εφαρμογές LP (Sewell 2005).

1.4.1 Το πρόβλημα κατανομής των πόρων

Ένα εργοστάσιο παράγει δύο προϊόντα: καρέκλες και πίνακες. Πραγματοποιούν ένα κέρδος $40 σε κάθε παραχθείσα καρέκλα και $50 σε κάθε πίνακα. Μια καρέκλα απαιτεί τους ακόλουθους πόρους για να παραχθεί: 2 ανθρωποώρες, 3 ώρες μηχανών και 1 μονάδα ξύλου. Ο πίνακας απαιτεί 2 ανθρωποώρες, 1 ώρα μηχανών και 4 μονάδες του ξύλου. Το εργοστάσιο έχει 60 ανθρωπώρες, 75 ώρες μηχανών, και 84 μονάδες ξύλου για κάθε ημέρα για την παραγωγή αυτών των δύο προϊόντων. Πώς θα έπρεπε οι πόροι (ανθρωποώρες, ώρες μηχανής, και ξύλο) να διατεθούν μεταξύ των δύο προϊόντων προκειμένου να μεγιστοποιηθεί το κέρδος του εργοστασίου;

Εάν θέσουμε x_1 τον αριθμό των καρεκλών που παράγονται ανά μέρα και x_2 τον αριθμό των πινάκων που παράγονται ανά μέρα, τότε το πρόβλημα μπορεί να τεθεί μαθηματικά ως εξής:

$$\text{Maximize } P = 40\,x_1 + 50\,x_2$$

Με τον περιορισμό ότι: $2\,x_1 + 2\,x_2 \leq 60,$

$3\,x_1 + x_2 \leq 75,$

$x_1 + 4\,x_2 \leq 84,$

Και τα όρια $x_1 \geq 0, x_2 \geq 0,$

Η συνάρτηση για να μεγιστοποιηθεί, $40x_1 + 50x_2$, αναπαριστά το κέρδος, και οι περιορισμοί δηλώνουν ότι οι συνολικές ανθρωποώρες, ώρες μηχανών και ξύλου που θα χρησιμοποιηθούν δεν μπορούν να υπερβούν τα διαθέσιμα. Τα όρια θέτουν τα προφανή γεγονότα ότι το εργοστάσιο δεν μπορεί να παραγάγει έναν αρνητικό αριθμό καρεκλών ή πινάκων.

1.5 Γιατί ο Αλγόριθμος Simplex?

Οι Έλληνες και οι Ρωμαίοι εφηύραν πολλούς επιστημονικούς και εφαρμοσμένης μηχανικής αλγορίθμους, αλλά θεωρείται ότι ο όρος "αλγόριθμος" προέρχεται από το όνομα του άραβα μαθηματικού Al- khwarizmi (9ος αιώνας), ο οποίος έγραψε το βιβλίο Al -Al-jabr wa'l muqabalach, το οποίο εξελίχθηκε τελικά στα σημερινά εγχειρίδια άλγεβρας γυμνασίου. Οι Dongarra και Sullivan θέτουν μαζί έναν κατάλογο δέκα κορυφαίων αλγορίθμων του εικοστού αιώνα (Dongarra and Sullivan 2000). Σύμφωνα με αυτούς τους συγγραφείς, οι ακόλουθοι αλγόριθμοι (που απαριθμούνται κατά τη χρονολογική σειρά) είχαν τη μέγιστη επιρροή στην επιστήμη και την εφαρμοσμένη μηχανική στο παρελθόν:

- **1946:** The Metropolis Algorithm for Monte Carlo.
- **1947:** The Simplex Method for Linear Programming.
- **1950:** The Krylov Subspace Iteration Methods.
- **1951**: The Decompositional Approach to Matrix Computations.
- **1957:** The Fortran Optimizing Compiler.
- **1959-1961:** The QR Algorithm for Computing Eigenvalues.
- **1962:** Quicksort Algorithm for Sorting.

- **1965:** Fast Fourier Transform.
- **1977:** Integer Relation Detection.
- **1987:** Fast Multipole Method.

Ο αλγόριθμος simplex του γραμμικού προγραμματισμού έχει ορισθεί σαν ένας από τους "10 αλγόριθμους με τη μέγιστη επιρροή στην ανάπτυξη και την πρακτική της επιστήμης και της εφαρμοσμένης μηχανικής στο 20ο αιώνα.

1.6 Γιατί ο παράλληλος Υπολογισμός?

Ο παράλληλος υπολογισμός ορίζεται ως η πρακτική της χρησιμοποίησης ενός μεγάλου αριθμού συνεργαζόμενων επεξεργαστών, που επικοινωνούν μεταξύ τους για να λύσουν γρήγορα τα μεγάλα προβλήματα. Ο παράλληλος προγραμματισμός εξελισσεται με μεγάλη ταχύτητα σε μια σημαντική περιοχή της πληροφορικής. Είναι δυνατό ώστε στα επόμενα έτη αυτή η περιοχή να έχει γίνει τόσο ευρεία και ισχυρή ώστε το μεγαλύτερο μέρος της έρευνας που πραγματοποιείται στους τομείς του σχεδιασμού και της ανάλυσης των αλγορίθμων, των γλωσσών υπολογιστών, των εφαρμογών υπολογιστών και των αρχιτεκτονικών υπολογιστών θα είναι μέσα στο πλαίσιο του παράλληλου υπολογισμού (Mousa 2005)

1.6.1 Η ανάγκη του παράλληλου Υπολογισμού

Δίνουμε το ακόλουθο παράδειγμα για να σκιαγραφήσουμε την ανάγκη για περισσότερη υπολογιστική δύναμη.

Υποθέτουμε ότι θέλουμε να προβλέψουμε τον καιρό στην Ευρώπη για τις επόμενες δύο ημέρες. Επίσης υποθέτουμε ότι θέλουμε να μοντελοποιήσουμε την ατμόσφαιρα από τη στάθμη θάλασσας μέχρι ένα ύψος 20 χιλιομέτρων, και πρέπει να κάνουμε μια πρόβλεψη του καιρού ανά ώρα για τις επόμενες ημέρες.

Μια τυποποιημένη προσέγγιση σε αυτόν τον τύπο προβλήματος είναι να καλυφθεί η περιοχή ενδιαφέροντος με ένα πλέγμα και έπειτα να προβλεφθεί ο καιρός σε κάθε κελί (vertex) του πλέγματος. Έτσι υποθέτουμε ότι χρησιμοποιούμε ένα κυβικό πλέγμα, με κάθε κύβο να έχει μήκος 0,1 χιλιόμετρο σε κάθε πλευρά. Δεδομένου ότι η περιοχή της Ευρώπης είναι περίπου 11 εκατομμύριο τετραγωνικά χιλιόμετρα, χρειαζόμαστε τουλάχιστον

$$11 \times 10^6 \times 20 \times 10^3 = 22 \times 10^{10} \text{ σημεία πλέγματος.} \quad (1.1)$$

Εάν χρειαστούν 100 υπολογισμοί για να καθοριστεί ο καιρός σε ένα χαρακτηριστικό σημείο του πλέγματος, κατόπιν προκειμένου να προβλεφθεί ο καιρός μια ώρα από τώρα, θα πρέπει να κάνουμε περίπου 22 × 1012 υπολογισμούς. Δεδομένου ότι θέλουμε να προβλέψουμε τον καιρό σε κάθε ώρα για 48 ώρες, πρέπει να κάνουμε συνολικά περίπου

$$22 \times 10^{12} \text{ calculations} \times 48 \text{ hours} \approx 10^{15} \text{ calculations} \quad (1.2)$$

Εάν ο υπολογιστής μας μπορεί να εκτελεί 10^9 υπολογισμούς ανά δευτερόλεπτο, θα διαρκέσει περίπου 10^{15} υπολογισμούς/ 10^9υπολογισμούς ανά δευτερόλεπτο = 10^6 δευτερόλεπτα ≈ 11 ημέρες ! (1.3)

Με άλλα λόγια, ο υπολογισμός είναι μάταιος εάν μπορούμε να διενεργήσουμε μόνο 109 διαδικασίες ανά δευτερόλεπτο. Εάν, αφ' ετέρου, μπορούμε να πραγματοποιήσουμε 1012 υπολογισμούς ανά δευτερόλεπτο, θα μας πάρει περίπου 16 λεπτά για να πραγματοποιήσει τους υπολογισμούς. Έτσι θα είμαστε πραγματικά σε θέση να κάνουμε μια πλήρη πρόβλεψη του καιρού κατά τη διάρκεια κάθε μιας από τις επόμενες 48 ώρες. Δεν είναι δύσκολο να φανταστούμε τις απλές τροποποιήσεις σε αυτό το πρόβλημα έτσι ώστε οι 109 διαδικασίες ανά δευτερόλεπτο να μην είναι ικανοποιητικές. Παραδείγματος χάριν, να αντικαταστήσουμε την Ευρώπη με ολόκληρη τη γη. Κατόπιν η περιοχή θα πήγαινε από 11×106 σε περίπου 5×108 τετραγωνικά χιλιόμετρα Έτσι ο απαραίτητος χρόνος υπολογισμού θα αυξανόταν από 16 λεπτά σε 12 ώρες περίπου και οι πρώτες 11 προβλέψεις μας θα ήταν άχρηστες. Επιπλέον, δεν είναι δύσκολο να βρούμε τα απολύτως διαφορετικά προβλήματα με απέραντα μεγαλύτερη υπολογιστική δύναμη από

αυτήν που κατέχουμε τώρα. Για παράδειγμα, οι λεπτομερείς ατομικού επιπέδου προσομοιώσεις των βιομορίων και οι πολυάριθμοι τύποι προσομοιώσεων που θα επέσπευδαν το σχέδιο και η κατασκευή των ολοκληρωμένων κυκλωμάτων απαιτούν απέραντα τη μεγαλύτερη υπολογιστική δύναμη από αυτή που εμείς κατέχουμε.

Είναι προφανές ότι ο μόνος τρόπος γύρω από το πρόβλημα αυτό είναι να χρησιμοποιήσουμε τον παραλληλισμό. Η ιδέα εδώ είναι ότι εάν αρκετές πράξεις εκτελούνται ταυτόχρονα, τότε ο απαιτούμενος για τον υπολογισμό χρόνος μπορεί να μειωθεί σημαντικά.

1.7 Οργάνωση της Διατριβής

Η διατριβή αποτελείται από 6 κεφάλαια τα οποία οργανώνονται ως εξής: Κέθε κεφάλαιο αρχίζει με εισαγωγή στην οποία αναφέρονται πληροφορίες και ιδέες για τα περιεχόμενα του κεφαλαίου. Στο επόμενο κεφάλαιο, εισάγουμε μερικές παραλλαγές Αλγορίθμων της μορφής Simplex, για παράδειγμα Πρωταρχικός Αλγόριθμος Simplex, Δυαδικός Αλγόριθμος Simplex, Αλγόριθμος Simplex Εξωτερικού Σημείου, Πρωταρχικός εξωτερικός Αλγόριθμος Simplex και Δυαδικός Αλγόριθμος Simplex εξωτερικού σημείου.

Στο κεφάλαιο τρία παρουσιάζουμε τον αλγόριθμο simplex και τον αλγόριθμο simplex εξωτερικού σημείου στην αντίστροφη μορφή. Αφού οι κανόνες περιστροφής περιγράφουν πώς επιλέγονται η είσοδος και η έξοδος των μεταβλητών, παρουσιάζουμε οκτώ διαφορετικούς τρόπους κανόνων περιστροφής. Επίσης παρουσιάζουμε δυο διαφορετικά σχήματα ανανέωσης αντίστροφης βάσης. Τα περισσότερο γνωστά σχήματα είναι: α) the Product Form of the Inverse (PFI), β) the Elimination Form of the Inverse (EFI), και γ) για μέθοδο που πρόσφατα αναπτύχθηκε από τον Benhamadou 2002 .

Στο τέταρτο κεφάλαιο, παρουσιάζουμε μια υπολογιστική μελέτη η οποία κάνει σύγκριση μεταξύ του αλγορίθμου simplex εξωτερικού σημείου EPSA και του κλασσικού πρωταρχικού αλγόριθμου simplex PSA . Στο πέμπτο κεφάλαιο παρουσιάζουμε έρευνα για τον παράλληλο υπολογισμό και παρουσιάζουμε τον δικό μας MPI παράλληλο κώδικα simplex.

Στο τελευταίο κεφάλαιο της διατριβής συζητούμε και συνοψίζουμε μερικά προβλήματα και προτάσεις για να επεκτείνουμε και γενικεύσουμε την εργασία τα διατριβής σε αρκετές κατευθύνσεις.

ΚΕΦΑΛΑΙΟ 2

ΑΛΓΟΡΙΘΜΟΙ ΤΥΠΟΥ-SIMPLEX

2.1 Εισαγωγή

Σ' αυτό το κεφάλαιο απαντάμε στην ερώτηση "ποια είναι η γεωμετρική ερμηνεία της μεθόδου Simplex"; Κατόπιν περιγράφουμε διάφορους τύπους αλγορίθμων Simplex π.χ. τον πρωτεύοντα αλγόριθμο Simplex, τον δυϊκό αλγόριθμο Simplex, τον αλγόριθμο Simplex εξωτερικών σημείων, τον πρωτεύοντα αλγόριθμο Simplex εξωτερικών σημείων και τον δυϊκό αλγόριθμο Simplex εξωτερικών σημείων.

Ο πρώτος αλγόριθμος Simplex εξωτερικών σημείων (EPSA) αναπτύχθηκε από τον Παπαρρίζο για το πρόβλημα της αντιστοίχισης (Paparrizos 1991). Αργότερα ο Παπαρρίζος γενίκευσε τον EPSA για την επίλυση του γενικευμένου γραμμικού προβλήματος LP (Paparrizos 1993). Πρωτεύουσες και δυϊκές εκδόσεις του αλγορίθμου περιγράφονται στις εργασίες (Paparrizos 1996; Paparrizos 1997). Από τη γεωμετρία του EPSA, (Paparrizos et al. 2000) είχε προβλεφθεί ότι οι νέοι αλγόριθμοι θα ήταν πιο γρήγοροι από τον αλγόριθμο Simplex. Αυτή η πρόβλεψη επιβεβαιώθηκε από υπολογιστικές μελέτες σε τυχαία αραιά και πυκνά γραμμικά προβλήματα (Paparrizos et al. 2000). Πιο συγκεκριμένα αποδείχθηκε ότι ο EPSA είναι μέχρι 10 φορές πιο γρήγορος από τον αλγόριθμο Simplex για τυχαία βέλτιστα γραμμικά προβλήματα μεσαίου μεγέθους.

Οι Anstreicher and Terlaky 1994, Curet 1984, Chen et al. 1974 και οι Murty and Fathi 1984 ανέπτυξαν άλλους αλγόριθμους τύπου Simplex, οι οποίοι με τον έναν ή τον άλλο τρόπο αποφεύγουν την εφικτή περιοχή. Μια πλήρης επισκόπηση των κανόνων περιστροφής του γραμμικού προγραμματισμού μπορούν να βρεθούν στην αναφορά (Terlaky and Zhang 1993).

2.2 Πως δουλεύει γεωμετρικά η μέθοδος Simplex;

Ας θεωρήσουμε το παρακάτω πρόβλημα γραμμικού προγραμματισμού:

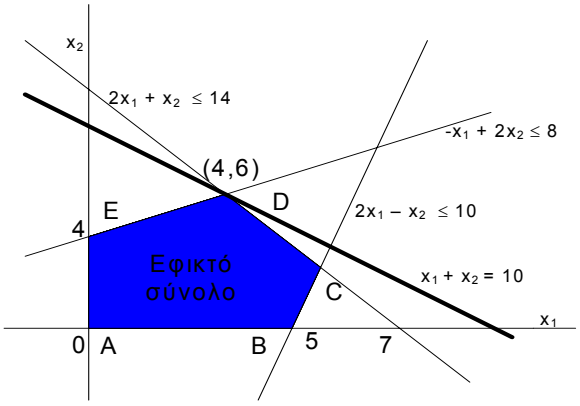
$$\begin{aligned} &\text{Μεγιστοποίησε} && x_1 + x_2 \\ &\text{Με περιορισμούς} && 2x_1 + x_2 \le 14 \\ & && -x_1 + 2x_2 \le 8 \\ & && 2x_1 - x_2 \le 10 \\ & && x_1 \ge 0,\ x_2 \ge 0 \end{aligned} \qquad (2.2.1)$$

Η μέθοδος Simplex αποτελείται από 2 βασικά βήματα που καλούνται «φάσεις». Στην πρώτη φάση προσπαθούμε να βρούμε μια εφικτή λύση του προβλήματος. Για προβλήματα μικρού μεγέθους ή ακόμη και για μεγαλύτερα προβλήματα κάποιων μορφών η παραπάνω λύση δεν είναι δύσκολη. Συχνά μια τετριμμένη λύση, όπως η $x_1 = 0$, είναι μια εφικτή λύση.

Αφού βρεθεί μια εφικτή λύση στο πρόβλημα, η μέθοδος Simplex προσπαθεί να βελτιώσει σε κάθε επανάληψη την τιμή της συνάρτησης κόστους (αντικειμενική συνάρτηση). Αυτό επιτυγχάνεται βρίσκοντας μια μεταβλητή του προβλήματος η τιμή της οποίας μπορεί να αυξηθεί, μειώνοντας ταυτόχρονα κάποια άλλη μεταβλητή έτσι ώστε να προκύπτει βελτίωση της τιμής της συνάρτησης κόστους. Αυτό μπορεί να οπτικοποιηθεί γραφικώς, μετακίνησης στις πλευρές του εφικτού (πολυγώνιου) από κορυφή σε κορυφή.

Το *εφικτό σύνολο* του προβλήματος 2.2.1 μπορεί να παρασταθεί σε 2 διαστάσεις όπως φαίνεται στο Σχήμα 2.2.1 Οι μη αρνητικοί περιορισμοί $x_1 \geq 0$ και $x_2 \geq 0$ περιορίζουν το εφικτό σύνολο στο πρώτο τεταρτημόριο. Οι υπόλοιποι 3 περιορισμοί είναι εξισώσεις ευθείας στο επίπεδο x_1- x_2 όπως φαίνεται στο σχήμα. Η συνάρτηση κόστους $x_1 + x_2$, μπορεί να αναπαρασταθεί με μια ευθεία γραμμή που έχει συντελεστή διεύθυνσης ίσο με –1. Η τεταγμένη της γραμμής της συνάρτησης κόστους ισούται με την τιμή της συνάρτησης κόστους για οποιαδήποτε λύση που βρίσκεται επάνω στη γραμμή. Η έντονη γραμμή του σχήματος 2.2.1 αναπαριστά τη βέλτιστη λύση του προβλήματος, διότι είναι μια γραμμή που έχει συντελεστή διεύθυνσης ίσο με –1 και μέγιστη τεταγμένη ίση με 10 που τέμνει το εφικτό σύνολο. Η τιμή της συνάρτησης κόστους για την εφικτή λύση ισούται με 10 και η ευθεία $x_1 + x_2 = 10$ της συνάρτησης κόστους έχει ακριβώς ένα σημείο με συντεταγμένες $x_1 = 4$ και $x_2 = 6$ μέσα στο εφικτό σύνολο.

Η μέθοδος Simplex προσπαθεί να βρει μια εφικτή λύση (ένα σημείο), και μετά μετακινεί αυτό το σημείο προς οποιαδήποτε κορυφή του εφικτού συνόλου, με στόχο τη βελτίωση της συνάρτησης κόστους. Τελικά φθάνουμε σε μια κορυφή από την οποία αν μετακινηθούμε δεν βελτιώνεται η συνάρτηση κόστους. Αυτή είναι και η βέλτιστη λύση. Στο παράδειγμά μας η τιμή $x_1 = 0$ και $x_2 = 0$ είναι μια τετριμμένη εφικτή λύση, που δίνει στη συνάρτηση κόστους την τιμή 0. Αυτή είναι η κορυφή Α του σχήματος 2.2.1. Από το σημείο αυτό μπορούμε να μετακινηθούμε είτε προς το σημείο Β είτε προς το Ε. Το σημείο Ε (0, 4) αυξάνει την τιμή της συνάρτησης κόστους σε 4 ενώ το σημείο Β (5, 0) την αυξάνει στην τιμή 5. Επειδή το σημείο Β μας δίνει μεγαλύτερη βελτίωση, το επιλέγουμε στην πρώτη επανάληψη (θα μπορούσαμε επίσης να είχαμε επιλέξει και το σημείο Ε, κάτι που θα μας οδηγούσε πιο γρήγορα στη βέλτιστη λύση). Η τιμή της μεταβλητής x_1 αυξάνει από 0 σε 5, ενώ η τιμή της x_2 παραμένει 0.



Σχήμα 2**.** 2**.** 1**.** Γραφική επίλυση ενός γραμμικού προβλήματος.

Από το σημείο B, ελέγχουμε αν μια μετακίνηση προς το σημείο C είναι επωφελής (γνωρίζουμε ότι μια ενδεχόμενη μετακίνηση προς το σημείο A δεν είναι επωφελής). Το σημείο C (6, 2) δίνει την τιμή 8 στη συνάρτηση κόστους, γεγονός που αποτελεί βελτίωση. Συνεπώς αυξάνουμε την τιμή της μεταβλητής x_1 από 5 σε 6 και επειδή ισχύουν οι περιορισμοί $2\ x_1 - x_2 \leq 10$ και $- x_1 + 2\ x_2 \leq 8$, πρέπει να αυξήσουμε και την τιμή της x_2 από 0 σε 2. Από το σημείο C, ελέγχουμε αν μια μετακίνηση προς το σημείο D βελτιώνει την κατάσταση. Για το σημείο D (4, 6) η τιμή της συνάρτηση κόστους ισούται με 10 και έτσι αποδεχόμαστε την μετακίνηση, αυξάνοντας την τιμή της x_2 από 2 σε 6. Αυτό σημαίνει ότι η τιμή της x_1 πρέπει να μειωθεί από 6 σε 4 λόγω των περιορισμών που ισχύουν. Τώρα επειδή μια μετακίνηση είτε προς το σημείο E (τιμή της συνάρτησης κόστους ίση με 4) είτε προς το σημείο C (τιμή της συνάρτησης κόστους ίση με 8) μειώνει την τρέχουσα τιμή της συνάρτησης κόστους, συμπεραίνουμε ότι το σημείο D είναι η βέλτιστη λύση του προβλήματος.

2.3 Πρωτεύων Αλγόριθμος Simplex

Ας θεωρήσουμε το παρακάτω γραμμικό πρόβλημα:

Μεγιστοποίησε $z = cx$

Με περιορισμούς $Ax = b$ 2.3.1

$x \geq 0$

Μια βασική εφικτή λύση στο παραπάνω πρόβλημα αντιστοιχεί σε ένα ακραίο σημείο της εφικτής περιοχής και χαρακτηρίζεται μαθηματικά από τον διαμερισμό του πίνακα A σε έναν μη-μοναδιαίο πίνακα βάσης B και στον πίνακα N που αποτελείται από τις μη βασικές στήλες. Δηλαδή,

$$A = (B: N) \qquad 2.3.2$$

Βασιζόμενοι σε αυτό το διαμερισμό, το γραμμικό σύστημα $Ax = b$ μπορεί να ξαναγραφεί ως

$$Bx_B + Nx_N = b \qquad 2.3.3$$

Αυτό απλοποιείται περαιτέρω στο σύστημα

$$x_B + B^{-1}Nx_N = B^{-1}b \qquad 2.3.4$$

και λύνοντας ως προς x_B, με άγνωστο το x_N, παίρνουμε

$$x_B = B^{-1}b - B^{-1}Nx_N \qquad 2.3.5$$

Τώρα θέτοντας $x_N = 0$, βλέπουμε ότι η εξίσωση 2.3.5 έχει ως αποτέλεσμα την
$x_B = B^{-1}b$. Η λύση

$$x=\begin{pmatrix} x_B \\ x_N \end{pmatrix}=\begin{pmatrix} B^{-1}b \\ 0 \end{pmatrix}$$

καλείται βασική λύση, το διάνυσμα x_B καλείται διάνυσμα των βασικών μεταβλητών, και το x_N καλείται διάνυσμα των μη βασικών μεταβλητών. Επιπροσθέτως αν $x_B = B^{-1} b \geq 0$, τότε η παρακάτω λύση

$$x=\begin{pmatrix} B^{-1}b \\ 0 \end{pmatrix}$$

καλείται βασική εφικτή λύση.

Τώρα ας θεωρήσουμε την αντικειμενική συνάρτηση $z = cx$. Διαμερίζοντας το διάνυσμα κόστους c σε βασικά και μη-βασικά μέρη π.χ. $c = (c_B, c_N)$, η αντικειμενική συνάρτηση γράφεται ως

$$z=c_B x_B+c_N x_N \qquad 2.3.6$$

Αντικαθιστώντας το x_B (εξίσωση 2.3.5) στην εξίσωση 2.3.6 παίρνουμε

$$z=c_B(B^{-1}b-B^{-1}Nx_N)+c_N x_N \qquad 2.3.7$$

Η παραπάνω εξίσωση μπορεί να γραφεί ως:

$$z=c_B B^{-1}b-(c_B B^{-1}N-c_N)x_N \qquad 2.3.8$$

Συνεπώς το z ισούται με τη σταθερά $c_B B^{-1}b$ μείον τον όρο $(c_B B^{-1}N-c_N)x_N$. Αν θέσουμε x_N = 0, η εξίσωση 2.3.8 μετατρέπεται στην $z=c_B B^{-1}b$. Η τελευταία είναι η αντικειμενική τιμή που αντιστοιχεί στην τρέχουσα βασική εφικτή λύση. Έτσι η τρέχουσα λύση ακραίου σημείου μπορεί να παρασταθεί σε κανονική μορφή όπως παρακάτω:

$$z=c_B B^{-1}b-(c_B B^{-1}N-c_N)x_N \qquad 2.3.9$$

$$x_B = B^{-1}b - B^{-1}Nx_N \qquad 2.3.10$$

Η τρέχουσα βασική εφικτή λύση δίνεται από τις παρακάτω εξισώσεις

$$z=c_B B^{-1}b \qquad 2.3.11$$

$$x=\begin{pmatrix} x_B \\ x_N \end{pmatrix}=\begin{pmatrix} B^{-1}b \\ 0 \end{pmatrix}\geq 0 \qquad 2.3.12$$

Αν τώρα αναδιατάξουμε τους όρους, έτσι ώστε όλες οι αναθεωρημένες μεταβλητές να βρίσκονται στο αριστερό μέρος της εξίσωσης ενώ οι σταθερές να βρίσκονται στο δεξιό μέρος παίρνουμε:

$$z+(c_B B^{-1}N-c_N)x_N=c_B B^{-1}b \qquad 2.3.13$$

$$x_B + B^{-1}Nx_N = B^{-1}b \qquad 2.3.14$$

Το simplex ταμπλό είναι απλά ένας πίνακας που χρησιμοποιείται για την αποθήκευση των συντελεστών των εξισώσεων 2.3.13 και 2.3.14 . Η επάνω γραμμή (γραμμή 0) του ταμπλό αποτελείται από τους συντελεστές της αντικειμενικής εξίσωσης 2.3.13 και το σώμα του ταμπλό (γραμμές 1 έως m) αποθηκεύει τους συντελεστές των εξισώσεων των περιορισμών (2.3.14). Η γενική μορφή του ταμπλό φαίνεται στον πίνακα 2.3.1 και αν ξαναγραφεί πιο συμπυκνωμένα προκύπτει η μορφή που φαίνεται στον πίνακα 2.3.2.

	z	x_B	x_N	Δεξί Μέρος	
z	1	0	$c_B B^{-1}N - c_N$	$c_B B^{-1}b$	γραμμή 0
x_B	0	I	$B^{-1}N$	$B^{-1}b$	γραμμές 1-m

Πίνακας 2.3.1

	z	x	Δεξί Μέρος	
z	1	$c_B B^{-1}A - c$	$c_B B^{-1}b$	γραμμή 0
x_B	0	$B^{-1}A$	$B^{-1}b$	γραμμές 1 -m

Πίνακας 2.3.2

Τώρα είμαστε έτοιμοι να συνοψίσουμε τα βήματα του αλγορίθμου Simplex όπως αυτά εφαρμόζονται στο simplex ταμπλό.

Περιγραφή του Πρωτεύοντος Αλγορίθμου Simplex

Βήμα0: (Αρχικοποίηση)

Ξεκίνα με ένα εφικτό βασικό σημείο και κατασκεύασε το αντίστοιχο simplex ταμπλό.

Βήμα1:(Επιλογή εισερχόμενης μεταβλητής)

Αν $a_{0j} \geq 0$ για j=1, 2, …, n, STOP. Η λύση είναι βέλτιστη. Με κάποιον κανόνα περιστροφής, επέλεξε την εισερχόμενη μεταβλητή με δείκτη s:

$a_{0s} > 0$, π.χ.

$$a_{0s} = \min\{ a_{0j} : \ a_{0j} < 0, \ \text{για} \ \ j = 1,2, \ldots, n\}$$

Βήμα2: **(Επιλογή εξερχόμενης μεταβλητής)**

Θέσε $I = \{ i : a_{is} > 0\}$. Αν $I=\varnothing$, STOP. Το πρόβλημα είναι απεριόριστο. Με κάποιον κανόνα περιστροφής, βρες το δείκτη της στήλης περιστροφής r:

$$\frac{b_r}{a_{rs}} = \min\left\{\frac{b_i}{a_{is}} : \ i \in I\right\}$$

Βήμα3: **(Περιστροφή)**

Σχημάτισε το επόμενο ταμπλό με τη μεταβλητή περιστροφής a_{rs} π.χ.

Θέσε $\quad a_{rj} \leftarrow \dfrac{a_{rj}}{a_{rs}}$ όπου j=1, 2, … ,n, n+1

$a_{ij} \leftarrow a_{ij} - \dfrac{a_{rj}}{a_{rs}} a_{is}$ όπου i= 0, 1, 2, . . . m $(i \neq r)$ και j = 1, 2, … , n, n+1

και πήγαινε στο βήμα1.

Με τον Πρωτεύοντα Αλγόριθμο Simplex επιλύουμε 2 προβλήματα. Το πρώτο πρόβλημα είναι βέλτιστο και το δεύτερο απεριόριστο.

Παράδειγμα 2.3.1:

Να λυθεί το παρακάτω γραμμικό πρόβλημα με τον πρωτεύοντα αλγόριθμο Simplex.

$$
\begin{aligned}
\max \quad z = \; & 2x_1 + 3x_2 \\
\text{s.t.} \quad & x_1 - 2x_2 + x_3 = 4 \\
& 2x_1 + x_2 + x_4 = 18 \\
& x_2 + x_5 = 10 \\
& x_1,\ x_2,\ x_3,\ x_4 \text{ και } x_5 \geq 0
\end{aligned}
$$

Λύση:

Επανάληψη 1

Βήμα0 (Αρχικοποίηση)

Από το αρχικό πρόβλημα είναι φανερό ότι τα βασικά εφικτά σημεία είναι τα $x_1 = 0$, $x_2 = 0$, $x_3 = 4$, $x_4 = 18$ και $x_5 = 10$. Έτσι προκύπτει το παρακάτω αρχικό simplex ταμπλό:

	z	x_1	x_2	x_3	x_4	x_5	Δεξί Μέρος
z	1	-2	-3	0	0	0	0
x_3	0	1	-2	1	0	0	4
x_4	0	2	1	0	1	0	18
x_5	0	0	1	0	0	1	10

Βήμα1: (επιλογή εισερχόμενης μεταβλητής)

Προσδιορισμός της στήλης περιστροφής και επιλογή εισερχόμενης μεταβλητής

$$a_{0s} = \min\{a_{01}, a_{02}\} = \min\{-2, -3\} = -3 = a_{02}$$

Επειδή s = 2, η μεταβλητή x_2 μπαίνει στη βάση.

Βήμα2: (επιλογή εξερχόμενης μεταβλητής). Θέσε I = {2, 3} και βρες το r = 3 διότι

$$\frac{b_r}{a_{r3}} = \min\left\{\frac{b_2}{a_{23}}, \frac{b_3}{a_{33}}\right\} = \min\left\{\frac{18}{1}, \frac{10}{1}\right\} = 10$$

Συνεπώς η μεταβλητή x_5 βγαίνει από τη βάση.

Βήμα3: (Περιστροφή)

Σχημάτισε το επόμενο simplex ταμπλό με το στοιχείο περιστροφής a_{32} π.χ.

	z	x_1	x_2	x_3	x_4	x_5	Δεξί Μέρος
z	1	-2	0	0	0	3	30
x_3	0	1	0	1	0	2	4
x_4	0	2	0	0	1	-1	8
x_2	0	0	1	0	0	1	10

Επειδή $a_{01} < 0$, στην αντικειμενική γραμμή, επαναλαμβάνουμε τον πρωτεύοντα αλγόριθμο Simplex.

Επανάληψη 2

Βήμα1: (επιλογή εισερχόμενης μεταβλητής)

Προσδιορισμός της στήλης περιστροφής. Επέλεξε $a_{0s} = \min\{a_{01}, a_{02}\} = \min\{-2, 0\} = -2 = a_{01}$

.Το s = 1, συνεπώς η μεταβλητή x_1 μπαίνει στη βάση.

Βήμα2: (επιλογή εξερχόμενης μεταβλητής). Θέσε $I = \{1, 2\}$ και βρες το r = 2 διότι

$$\frac{b_r}{a_{r1}} = \min\left\{\frac{b_1}{a_{13}}, \frac{b_2}{a_{23}}\right\} = \min\left\{\frac{24}{1}, \frac{8}{1}\right\} = 4$$

Συνεπώς η μεταβλητή x_4 βγαίνει από τη βάση.

Βήμα3: (Περιστροφή)

Σχημάτισε το επόμενο simplex ταμπλό με το στοιχείο περιστροφής a_{21} π.χ.

	z	x_1	x_2	x_3	x_4	x_5	Δεξί Μέρος
z	1	0	0	0	1	2	38
x_3	0	0	0	1	-1/2	5/2	20
x_1	0	1	0	0	1/2	-1/2	4
x_2	0	0	1	0	0	1	10

Επειδή $a_{0j} \geq 0$, όπου j =1, 2, . . ., 5 στην αντικειμενική γραμμή STOP. Η λύση είναι βέλτιστη και είναι η $x_1 = 4$, $x_2 = 10$, $x_3 = 20$, $x_4 = 0$, $x_5 = 0$. Η αντικειμενική συνάρτηση είναι $z = 2\ x_1 + 3\ x_2 = 38$.

Παράδειγμα 2.3.2

Να λυθεί το παρακάτω γραμμικό πρόβλημα με τον πρωτεύοντα αλγόριθμο Simplex.

$$\begin{aligned} \max.\ z = \ & 5x_1 + 3x_2 \\ \text{s.t.} \quad & -x_1 + x_2 \leq 4 \\ & x_1 - 2x_2 \leq 6 \\ & x_1 \text{ και } x_2 \geq 0 \end{aligned}$$

Λύση:

Προσθέτουμε τις χαλαρές μεταβλητές x_3 και x_4 έτσι ώστε το γραμμικό πρόβλημα να μετατραπεί στο παρακάτω:

$$\begin{aligned} \max \quad z = \ & 5x_1 + 3x_2 \\ \text{s.t.} \quad & -x_1 + x_2 + x_3 = 4 \\ & x_1 - 2x_2 + x_4 = 6 \\ & x_1, x_2, x_3 \text{ και } x_4 \geq 0 \end{aligned}$$

Επανάληψη1

Βήμα0 (Αρχικοποίηση)

Σχημάτισε το αρχικό πρόβλημα. Είναι φανερό ότι τα βασικά εφικτά σημεία είναι τα $x_1 = 0$, $x_2 = 0$, $x_3 = 4$ και $x_4 = 6$. Έτσι κατασκευάζουμε το αρχικό simplex ταμπλό

	z	x_1	x_2	x_3	x_4	Δεξί Μέρος
z	1	-5	-3	0	0	0
x_3	0	-1	1	1	0	4
x_4	0	1	-2	0	1	6

Βήμα1: (επιλογή εισερχόμενης μεταβλητής)

Προσδιόρισε τη στήλη περιστροφής και επέλεξε

$$a_{0s} = \min\{a_{01}, a_{02}\} = \min\{-5, -3\} = -5 = a_{01}$$

Το s = 1, συνεπώς η μεταβλητή x_1 μπαίνει στη βάση.

Βήμα2: (επιλογή εξερχόμενης μεταβλητής). Θέσε I = {2} και βρες το r = 2 διότι

$$\frac{b_r}{a_{r1}} = \min\left\{\frac{b_2}{a_{21}}\right\} = \min\left\{\frac{6}{1}\right\} = 6$$

Συνεπώς η μεταβλητή x_4 βγαίνει από τη βάση.

Βήμα3: (Περιστροφή)

Σχημάτισε το επόμενο simplex ταμπλό με το στοιχείο περιστροφής a_{21} π.χ.

	z	x_1	x_2	x_3	x_4	Δεξί Μέρος
z	1	0	-13	0	5	30
x_3	0	0	-1	1	1	10
x_1	0	1	-2	0	1	6

Επειδή υπάρχει το $a_{02} < 0$, στην αντικειμενική γραμμή, επαναλαμβάνουμε τον πρωτεύοντα αλγόριθμο simplex

Επανάληψη2

Βήμα1: (επιλογή εισερχόμενης μεταβλητής).

Προσδιόρισε τη στήλη περιστροφής και επέλεξε

$$a_{0s} = \min\{ a_{02}\} = \min\{-13\} = -13 = a_{02}$$

Το s = 2, συνεπώς η μεταβλητή x_2 μπαίνει στη βάση.

Βήμα2: (επιλογή εξερχόμενης μεταβλητής) Θέσε $I = \varnothing$ έτσι ώστε το πρόβλημα να είναι απεριόριστο.

2.4 Δυϊκός Αλγόριθμος Simplex

Σε αυτή την ενότητα παρουσιάζουμε ένα από τα πιο σημαντικά ζητήματα του γραμμικού προγραμματισμού. Κατά την εκτέλεση του αλγορίθμου Simplex δεν βρίσκουμε μόνο τη βέλτιστη λύση ενός γραμμικού προβλήματος αλλά βρίσκουμε επίσης και ένα συνοδευτικό πρόβλημα που καλείται δυϊκό πρόβλημα. Η δυϊκή μέθοδος Simplex αναπτύχθηκε από τον Lemke 1954. Στον γραμμικό προγραμματισμό χρησιμοποιούμε τη δυϊκότητα σε ένα μεγάλο πλήθος θεωρητικών και πρακτικών περιπτώσεων. Μεταξύ άλλων αυτές είναι οι παρακάτω:

1- Σε μερικές περιπτώσεις είναι πιο εύκολο (π.χ. λιγότερες επαναλήψεις) να λύσουμε το δυϊκό πρόβλημα αντί του πρωτεύοντος.

2- Οι δυϊκές μεταβλητές παρέχουν σημαντικές οικονομικές ερμηνείες όταν επιλύουμε ένα πρόβλημα γραμμικού προγραμματισμού.

3- Η δυϊκότητα χρησιμοποιείται βοηθητικά όταν αλλάζουν οι τιμές των συντελεστών ενός προβλήματος γραμμικού προγραμματισμού.

4- Η δυϊκότητα χρησιμοποιείται για να μας επιτρέψει να χρησιμοποιήσουμε τη μέθοδο simplex για να επιλύσουμε προβλήματα στα οποία η αρχική βάση δεν είναι εφικτή (αυτή η τεχνική είναι γνωστή σαν δυϊκή simplex).

5- Η δυϊκότητα χρησιμοποιείται για να αναπτύξουμε έναν αριθμό από σημαντικά θεωρητικά αποτελέσματα στο γραμμικό προγραμματισμό.

Περιγράφουμε τη δυϊκή μέθοδο Simplex η οποία λύνει το δυϊκό πρόβλημα απευθείας πάνω στο simplex ταμπλό του πρωτεύοντος. Εξετάζουμε το σύνηθες πρόβλημα του γραμμικού προγραμματισμού:

$$\begin{aligned} \max \quad & z = cx \\ \text{s.t.} \quad & Ax = \\ & x \geq 0 \end{aligned} \qquad 2.4.1$$

Σε κάθε επανάληψη προχωράμε από μια βασική εφικτή λύση του δυϊκού προβλήματος σε μια βελτιωμένη βασική εφικτή λύση μέχρις ότου είτε φτάσουμε σε μια βέλτιστη λύση του δυϊκού (καθώς επίσης και του πρωτεύοντος) είτε συμπεράνουμε ότι το δυϊκό πρόβλημα είναι απεριόριστο και το πρωτεύων είναι μη εφικτό.

Περιγραφή του Δυϊκού Αλγορίθμου Simplex

Βήμα0: (Αρχικοποίηση)

Ξεκίνα με ένα βασικό δυϊκό εφικτό σημείο και κατασκεύασε το αντίστοιχο simplex ταμπλό.

Βήμα1: (Επιλογή εξερχόμενης μεταβλητής)

Αν $b_i \geq 0$ για i =1, 2, …, m, STOP. Η λύση είναι βέλτιστη.

Διαφορετικά διάλεξε την εξερχόμενη μεταβλητή του δείκτη r έτσι ώστε :

$$b_r = \min\{ b_i : \ b_i < 0, \ \text{για} \ \ i = 1,2, \ldots, m\}$$

Βήμα2: **(Επιλογή εισερχόμενης μεταβλητής)**

Θέσε $J = \{ j : a_{rj} < 0\}$. Αν $J = \varnothing$, STOP. Το δυϊκό πρόβλημα είναι απεριόριστο. Διαφορετικά βρες τον δείκτη της στήλης περιστροφής s :

$$\frac{a_{0s}}{a_{rs}} = \min\left\{\frac{-a_{0j}}{a_{rj}} : \ j \in J\right\}$$

Βήμα3: **(Περιστροφή)**

Από το επόμενο ταμπλό της μεταβλητής περιστροφής a_{rs} π.χ.

Θέσε $a_{rj} \leftarrow \frac{a_{rj}}{a_{rs}}$ όπου j=1, 2, … ,n, n+1

$a_{ij} \leftarrow a_{ij} - \frac{a_{rj}}{a_{rs}} a_{is}$ όπου i= 0, 1, 2, . . . m $(i \neq r)$ και j = 1, 2, … , n, n+1

και πήγαινε στο Βήμα1.

Επιλύουμε 2 προβλήματα με τον Δυϊκό Πρωτεύοντα Αλγόριθμο Simplex. Το πρώτο πρόβλημα είναι βέλτιστο και το δεύτερο απεριόριστο.

Παράδειγμα 2.4.1:

Να λυθεί το παρακάτω γραμμικό πρόβλημα με τον δυϊκό πρωτεύοντα αλγόριθμο Simplex.

$$\begin{aligned} \max \ z = & -2x_1 - x_2 - 4x_3 \\ \text{s.t.} \quad & -2x_1 - 4x_2 - 2x_3 \leq -6 \\ & -x_1 + 2x_2 - 6x_3 \leq -5 \\ & -x_1 + 4x_2 + 3x_3 \leq 8 \\ & x_1, \ x_2, \ x_3 \text{ και } x_4 \geq 0 \end{aligned}$$

Λύση:

Προσθέτουμε τις χαλαρές μεταβλητές x_4, x_5 και x_6, οπότε το γραμμικό πρόβλημα μετατρέπεται στο παρακάτω:

$$\begin{aligned} \max\ z = & -2x_1 - x_2 - 4x_3 \\ \text{s.t.} \quad & -2x_1 - 4x_2 - 2x_3 + x_4 = -6 \\ & -x_1 + 2x_2 - 6x_3 + x_5 = -5 \\ & -x_1 + 4x_2 + 3x_3 + x_6 = 8 \\ & x_1,\ x_2,\ x_3,\ x_4,\ x_5 \text{ και } x_6 \geq 0 \end{aligned}$$

Επανάληψη1

Βήμα0 (Αρχικοποίηση)

Από το αρχικό πρόβλημα είναι φανερό ότι τα δυϊκά βασικά εφικτά σημεία είναι τα $x_1 = 0$, $x_2 = 0$, $x_3 = 0$, $x_4 = -6$, $x_5 = -5$ και $x_6 = 8$. Κατασκευάζουμε το αρχικό simplex ταμπλό

	z	x_1	x_2	x_3	x_4	x_5	x_6	Δεξί Μέρος
z	1	2	1	4	0	0	0	0
x_4	0	-2	-4	-2	1	0	0	-6
x_5	0	-1	2	-6	0	1	0	-5
x_6	0	-1	4	3	0	0	1	8

Βήμα1: (επιλογή εξερχόμενης μεταβλητής)

Προσδιόρισε τη γραμμή περιστροφής και επέλεξε

$$b_r = \min\{b_1, b_2\} = \min\{-6, -5\} = -6 = b_1$$

Το r=1, συνεπώς η μεταβλητή x_4 βγαίνει από τη βάση.

Βήμα2: (επιλογή εισερχόμενης μεταβλητής) θέσε J = {1, 2, 3} και βρες s = 2 διότι

$$\frac{a_{0s}}{a_{rs}} = \min\left\{\frac{-a_{0j}}{a_{rj}} : j \in J\right\} = \min\left\{\frac{-a_{01}}{a_{11}}, \frac{-a_{02}}{a_{12}}, \frac{-a_{03}}{a_{13}}\right\} = \min\left\{\frac{-2}{-2}, \frac{-1}{-4}, \frac{-4}{-2}\right\} = \frac{1}{4}$$

Συνεπώς η μεταβλητή x_2 μπαίνει στη βάση.

Βήμα3: (Περιστροφή)

Σχημάτισε το επόμενο simplex ταμπλό με το στοιχείο περιστροφής

$a_{32} = a_{12} = -4$ π.χ.

	z	x_1	x_2	x_3	x_4	x_5	x_6	Δεξί Μέρος
z	1	3/2	0	7/2	1/4	0	0	-3/2
x_2	0	1/2	1	1/2	-1/4	0	0	3/2
x_5	0	-4	0	-14	1	2	0	-16
x_6	0	-3	0	1	1	0	1	2

Επειδή υπάρχει $b_2 < 0$, στο δεξί μέρος, επαναλαμβάνουμε τον δυϊκό αλγόριθμο simplex

Επανάληψη2

Βήμα1: (επιλογή εξερχόμενης μεταβλητής)

Προσδιόρισε τη στήλη περιστροφής και επέλεξε

$$b_r = \min\{b_2\} = \min\{-16\} = -16 = b_2$$

Το r = 2, συνεπώς η μεταβλητή x_5 βγαίνει από τη βάση.

Βήμα2: (επιλογή εισερχόμενης μεταβλητής) θέσε $J = \{1, 3\}$ και βρες s = 3 διότι

$$\frac{a_{0s}}{a_{rs}} = \min\left\{\frac{-a_{0j}}{a_{rj}} : j \in J\right\} = \min\left\{\frac{-a_{01}}{a_{21}}, \frac{-a_{03}}{a_{23}}\right\} = \min\left\{\frac{-6}{-4}, \frac{-14}{-14}\right\} = 1$$

Συνεπώς η μεταβλητή x_3 μπαίνει στη βάση.

Βήμα3: (Περιστροφή)

Σχημάτισε το επόμενο simplex ταμπλό με το στοιχείο περιστροφής

$a_{32}= {}_{23} = -14$ π.χ.

	z	x_1	x_2	x_3	x_4	x_5	x_6	Δεξί Μέρος
z	1	1/2	0	0	1/2	1/2	0	-11/2
x_2	0	5/14	1	1/20	-3/14	-1/14	0	13/14
x_3	0	2/7	0	1	-1/14	-1/7	0	8/7
x_6	0	-23/7	0	0	15/14	8/7	1	26/7

Βήμα1: (επιλογή εξερχόμενης μεταβλητής)

Επειδή $b_i > 0$ όπου i = 1, 2, 3 STOP. Η λύση είναι η $x_1 = 0$, $x_2 = 13/14$, $x_3 = 8/7$, $x_4 = x_5 = 0$ και το z = -11/2.

Example 2.4.2:

Να λυθεί το παρακάτω γραμμικό πρόβλημα με τον δυϊκό πρωτεύοντα αλγόριθμο Simplex.

$$\begin{aligned} \max\ z = \quad & -x_1 + 2x_2 + 4x_3 \\ \text{s.t.} \quad & x_1 + x_2 - 2x_3 \leq -4 \\ & 2x_1 - x_2 - 2x_3 \leq -2 \\ & x_1 - x_2 + 2x_3 \leq -8 \\ & x_1,\ x_2, \text{ και } x_3 \geq 0 \end{aligned}$$

Λύση:

Προσθέτουμε τις χαλαρές μεταβλητές x_4, x_5 και x_6. Έτσι το γραμμικό πρόβλημα μετατρέπεται στο παρακάτω:

$$
\begin{aligned}
\max\ z = & -x_1 + 2x_2 + 4x_3 & & & & & & & \\
\text{s.t.} \quad & x_1 + x_2 - 2x_3 + x_4 & & & & & = & -4 \\
& 2x_1 - x_2 - 2x_3 & & + x_5 & & & = & -2 \\
& x_1 - x_2 + 2x_3 & & & & + x_6 & = & -8
\end{aligned}
$$

$$x_1,\quad x_2,\quad x_3,\quad x_4, x_5 \text{ και } x_6 \quad \geq \quad 0$$

Επανάληψη 1

Βήμα0 (Αρχικοποίηση) Από το αρχικό πρόβλημα είναι φανερό ότι τα Δυϊκά

Βασικά Εφικτά Σημεία είναι τα $x_1 = 0$, $x_2 = 0$, $x_3 = 0$, $x_4 = -4$, $x_5 = -2$ και $x_6 = -8$. Κατασκευάζουμε το αρχικό simplex ταμπλό

	z	x_1	x_2	x_3	x_4	x_5	x_6	R.H.S
z	1	1	-2	4	0	0	0	0
x_4	0	1	1	-2	1	0	0	-4
x_5	0	2	-1	-2	0	1	0	-2
x_6	0	1	-1	2	0	0	1	-8

Βήμα1: (επιλογή εξερχόμενης μεταβλητής).

Προσδιόρισε τη γραμμή περιστροφής και επέλεξε

$$b_r = \min\{b_1, b_2, b_3\} = \min\{-4, -2, -8\} = -8 = b_3$$

Το r = 3, συνεπώς η μεταβλητή x_6 βγαίνει από τη βάση.

Βήμα2: (επιλογή εισερχόμενης μεταβλητής)

θέσε J = { 2} και βρες s = 2 διότι

$$\frac{a_{0s}}{a_{rs}} = \min\left\{\frac{-a_{0j}}{a_{rj}} : j \in J\right\} = \min\left\{\frac{-a_{02}}{a_{32}}\right\} = \min\left\{\frac{-(-2)}{-1}\right\} = -2$$

Συνεπώς η μεταβλητή x_2 μπαίνει στη βάση.

Βήμα3: (Περιστροφή)

Σχημάτισε το επόμενο simplex ταμπλό με το στοιχείο περιστροφής

$a_{rs} = a_{12} = -1$ π.χ.

	z	x_1	x_2	x_3	x_4	x_5	x_6	Δεξί Μέρος
z	1	-1	0	-5	0	0	-2	16
x_4	0	2	0	0	1	0	0	-12
x_5	0	1	0	-4	0	1	0	6
x_2	0	1	1	-2	0	0	1	8

Επειδή υπάρχει $b_1 < 0$ στο δεξί μέρος επαναλαμβάνουμε τον δυϊκό αλγόριθμο simplex

Επανάληψη 2

Βήμα1: (επιλογή εξερχόμενης μεταβλητής)

Προσδιόρισε τη γραμμή περιστροφής και επέλεξε

$$b_r = \min\{b_1\} = \min\{-12\} = -12 = b_1$$

Το r = 1, συνεπώς η μεταβλητή x_4 βγαίνει από τη βάση.

Βήμα2: (επιλογή εισερχόμενης μεταβλητής)

Θέσε $J = \varnothing$, STOP. Το δυϊκό πρόβλημα είναι απεριόριστο.

2.5 Αλγόριθμος Simplex Εξωτερικών Σημείων

Ο κλασικός αλγόριθμος Simplex είναι ο πιο διαδεδομένος αλγόριθμος του γραμμικού προγραμματισμού. Στην πράξη είναι αρκετά αποτελεσματικός για γραμμικά προβλήματα μικρού ή μεσαίου μεγέθους. Ο Borgwardt απέδειξε ότι ο αναμενόμενος αριθμός των επαναλήψεων για την επίλυση ενός γραμμικού προβλήματος (LP) με τον αλγόριθμο Simplex είναι πολυωνυμικός (Borgwardt 1982). Οι Klee–Minty απέδειξαν ότι η πολυπλοκότητα χειρότερης περίπτωσης είναι εκθετική (Klee–Minty 1972). Από τότε έχουν γίνει πολλές προσπάθειες για να βρεθούν καλύτεροι (πιο γρήγοροι) τρόποι ώστε να επιλυθούν γραμμικά προβλήματα. Έρευνες οδήγησαν στη ανάπτυξη αποτελεσματικών μεθόδων εσωτερικών σημείων (Interior Point Methods) (Gondzio 1996; Lustig et al 1992). Οι τελευταίες είναι πιο αποτελεσματικές από τον αλγόριθμο Simplex για μεγάλα γραμμικά προβλήματα. Παρόλα αυτά η υπολογιστική βελτίωση των αλγορίθμων περιστροφής παραμένει θέμα μεγάλης σπουδαιότητας. Ένας τρόπος για να βελτιωθεί ο αλγόριθμος simplex είναι να αποφευχθεί η μετακίνηση από μια κορυφή σε μια γειτονική της. Αυτό μπορεί να επιτευχθεί αποφεύγοντας τα όρια του πολύεδρου $P = \{x : Ax \leq b, x \geq 0\}$ και κατασκευάζοντας βασικές λύσεις που δεν είναι εφικτές. Ένας τέτοιος αλγόριθμος ονομάζεται αλγόριθμος simplex εξωτερικών σημείων (EPSA). Ο πρώτος EPSA αναπτύχθηκε από τον Παπαρρίζο για το πρόβλημα της αντιστοίχισης (Paparrizos 1991). Αργότερα ο Παπαρρίζος γενίκευσε τον EPSA για το γενικευμένο γραμμικό πρόβλημα (Paparrizos 1993). Πρωτεύουσες-Δυϊκές εκδόσεις του αλγορίθμου εξετάζονται στο άρθρο (Paparrizos 1996; Paparrizos 1997). Από τη γεωμετρία του EPSA, ο Παπαρρίζος προέβλεψε (Paparrizos et al. 2000) ότι ο νέος αλγόριθμος θα ήταν πιο γρήγορος από τον αλγόριθμο simplex. Αυτή η πρόβλεψη επαληθεύτηκε από υπολογιστικές μελέτες σε τυχαία αραιά και πυκνά γραμμικά προβλήματα (Paparrizos et al. 2000). Πιο συγκεκριμένα ο EPSA αποδείχθηκε ότι είναι έως και 10 φορές πιο γρήγορος από τον αλγόριθμο simplex για τυχαία βέλτιστα γραμμικά προβλήματα μεσαίου μεγέθους. Οι

Anstreicher and Terlaky 1994, Curet 1984, Chen et al. 1974 και οι Murty and Fathi 1984 ανέπτυξαν άλλους αλγορίθμους τύπου simplex, που με τον ένα ή τον άλλο τρόπο αποφεύγουν την εφικτή περιοχή. Μια πλήρης επισκόπηση των κανόνων περιστροφής του γραμμικού προγραμματισμού μπορεί να βρεθεί στο άρθρο (Terlaky and Zhang 1993).

Ο EPSA κατασκευάζει 2 μονοπάτια (διαδρομές) προς τη βέλτιστη λύση. Η πρώτη διαδρομή αποτελείται από βασικές αλλά όχι εφικτές λύσεις και ονομάζεται εξωτερική διαδρομή (exterior path). Η δεύτερη διαδρομή είναι εφικτή. Αποτελείται από ευθύγραμμα τμήματα τα άκρα των οποίων βρίσκονται μέσα στην εφικτή περιοχή. Ο EPSA στηρίζεται στην ιδέα ότι ακολουθώντας βήματα προς κατευθύνσεις που είναι συνδυασμοί δελεαστικών κατηφορικών διαδρομών, μπορεί να οδηγηθούμε σε σύγκλιση πιο γρήγορα από ότι με τον κλασικό αλγόριθμο simplex. Αν και ο EPSA είναι σαφώς αποδοτικότερος από τον κλασικό αλγόριθμο simplex (σε τυχαία πυκνά και αραιά γραμμικά προβλήματα) παρουσιάζει 2 υπολογιστικά μειονεκτήματα (Paparrizos et al. 2000). Πρώτον είναι δύσκολο να κατασκευάσουμε καλές κατευθύνσεις μετακίνησης. Χρησιμοποιούμε τον όρο « καλές κατευθύνσεις μετακίνησης » καταχρηστικά. Μια καλή κατεύθυνση μετακίνησης είναι μια κατεύθυνση που κάνει τον αλγόριθμο αποδοτικό στην πράξη. Γεωμετρικά μια καλή κατεύθυνση μετακίνησης είναι μια κατεύθυνση που οδηγεί πολύ κοντά στη βέλτιστη λύση. Στην πράξη οι 2 παραπάνω διαδρομές εξαρτώνται από το αρχικό εφικτό τμήμα και από την αρχική εφικτή κορυφή. Δεύτερον δεν υπάρχει γνωστός τρόπος να μετακινηθούμε μέσα στο εσωτερικό της εφικτής περιοχής. Αυτή η μετακίνηση θα παρέχει μεγαλύτερη ευελιξία κατά την αναζήτηση υπολογιστικά αποδοτικών κατευθύνσεων.

2.6 Πρωτεύων Αλγόριθμος Simplex Εξωτερικών Σημείων

Περιγράφουμε τον πρωτεύοντα αλγόριθμο simplex εξωτερικών σημείων (EPSA), που λύνει το πρόβλημα απευθείας πάνω στο (πρωτεύον) simplex ταμπλό. Εξετάζουμε το κλασικό πρόβλημα του γραμμικού προγραμματισμού :

$$\begin{aligned} \max \quad & z = cx \\ s.t. \quad & Ax = b \\ & x \geq 0 \end{aligned} \qquad 2.6.1$$

Ο αλγόριθμος ξεκινά με ένα βασικό εφικτό σημείο. Βρίσκουμε το σύνολο των μη-βασικών δεικτών $J_- = \{j : a_{0j} < 0\}$. Αν $J_- = \varnothing$, τότε το πρώτο κύριο σημείο είναι το πιο βέλτιστο και ο αλγόριθμος σταματάει. Διαφορετικά υπολογίζουμε το άθροισμα $a_{i0} = \sum_{j \in J_-} a_{ij}$ όπου i = 0,1, 2, . . ., m και βρίσκουμε το σύνολο των δεικτών $I_+ = \{i : a_{i0} > 0\}$. Αν $I_+ = \varnothing$, τότε το πρόβλημα είναι απεριόριστο. Διαφορετικά βρίσκουμε το δείκτη της εισερχόμενης μεταβλητής r από το κλάσμα $\frac{b_r}{a_{r0}} = \min\left\{\frac{b_i}{a_{i0}} : i \in I_+\right\}$. Για να βρούμε το δείκτη της εξερχόμενης μεταβλητής s , πρέπει να βρούμε το σύνολο των δεικτών $J_+ = \{j : a_{0j} > 0\}$ και μετά να υπολογίσουμε τα θ_1 και θ_2. Αν $\theta_1 \leq \theta_2$ θέσε s = k αλλιώς s = l. Το τελευταίο βήμα είναι η περιστροφή με το στοιχείο περιστροφής a_{rs}.

Ο EPSA κατασκευάζει 2 διαδρομές προς τη βέλτιστη λύση. Η πρώτη διαδρομή αποτελείται από βασικές αλλά μη εφικτές λύσεις και είναι στην ουσία μια εξωτερική διαδρομή (exterior path). Η δεύτερη διαδρομή είναι εφικτή. Αποτελείται από ευθύγραμμα τμήματα τα άκρα των οποίων βρίσκονται μέσα στα όρια της εφικτής περιοχής.

Περιγραφή του Αλγορίθμου Simplex Εξωτερικών Σημείων

Βήμα0: (Αρχικοποίηση)

Ξεκίνα με ένα εφικτό βασικό σημείο και κατασκεύασε το αντίστοιχο ταμπλό του εξωτερικού simplex.

Βήμα1: (Έλεγχος τερματισμού)

Βρες το σύνολο $J_- = \{j : a_{0j} < 0\}$. Αν $J_- = \varnothing$,STOP. Το πρόβλημα είναι βέλτιστο.

Διαφορετικά υπολόγισε το $a_{i0} = \sum_{j \in J_-} a_{ij}$ όπου i = 0,1, 2, . . ., m.

Βήμα2: **(Επιλογή εισερχόμενης μεταβλητής)**

Βρες το σύνολο $I_+ = \{i : a_{i0} > 0\}$. Αν $I_+ = \varnothing$, STOP. Το πρόβλημα είναι απεριόριστο.

Διαφορετικά υπολόγισε το δείκτη της εισερχόμενης μεταβλητής r από τη σχέση

$$\frac{b_r}{a_{r0}} = \min\left\{\frac{b_i}{a_{i0}} : i \in I_+\right\}$$

Βήμα3: **(Επιλογή εξερχόμενης μεταβλητής)**

Θέσε $J_+ = \{j : a_{0j} > 0\}$ και υπολόγισε

$$\theta_1 = \frac{-a_{0k}}{a_{rk}} = \min\left\{\frac{-a_{0j}}{a_{rj}} : j \in J_-, a_{rj} > 0\right\}$$

$$\theta_2 = \frac{-a_{0l}}{a_{rl}} = \min\left\{\frac{-a_{0j}}{a_{rj}} : j \in J_+, a_{rj} < 0\right\}$$

Βρες το δείκτη της εξερχόμενης μεταβλητής s, αν $\theta_1 \le \theta_2$ θέσε s = k αλλιώς s = l.

Βήμα4: **(Περιστροφή)**

Κατασκεύασε το επόμενο ταμπλό με το στοιχείο περιστροφής a_{rs} π.χ.

Θέσε $a_{rj} \leftarrow \frac{a_{rj}}{a_{rs}}$ όπου j=1, 2, … ,n, n+1

$a_{ij} \leftarrow a_{ij} - \frac{a_{rj}}{a_{rs}} a_{is}$ όπου i= 0, 1, 2, . . . m $(i \neq r)$ και j = 1, 2, … , n, n+1

Αν s = k θέσε $a_{0r} \leftarrow a_{0r} - 1$.

και πήγαινε στο βήμα 1.

Παράδειγμα 2.6.1

Να λυθεί το παρακάτω γραμμικό πρόβλημα με τον αλγόριθμο Simplex εξωτερικών σημείων.

$$\begin{aligned} \max\ z= \quad & -x_1 - x_2 + 4x_3 \\ \text{s.t.} \quad & x_1 + x_2 + x_3 \le 9 \\ & x_1 + x_2 - x_3 \le 2 \\ & -x_1 + x_2 + x_3 \le 4 \end{aligned}$$

$$x_1, x_2 \text{ και } x_3 \ge 0$$

Λύση:

Προσθέτουμε τις χαλαρές μεταβλητές x_4, x_5 και x_6, οπότε το γραμμικό πρόβλημα μετατρέπεται στο παρακάτω:

$$\begin{aligned} \max z = \quad & -x_1 - x_2 + 4x_3 \\ \text{s.t.} \quad & x_1 + x_2 + x_3 + x_4 = 9 \\ & x_1 + x_2 - x_3 + x_5 = 2 \\ & -x_1 + x_2 + x_3 + x_6 = 4 \end{aligned}$$

$$x_1, x_2 , x_3, x_4, x_5 \text{ και } x_6 \ge 0$$

Βήμα0 (Αρχικοποίηση)

Από το αρχικό πρόβλημα κατασκευάζουμε το αρχικό εξωτερικό simplex ταμπλό

		x_1	x_2	x_3	x_4	x_5	x_6	Δεξί Μέρος
z	-4	1	1	-4	0	0	0	0
x_4	2	1	1	2	1	0	0	9
x_5	-1	1	1	-1	0	1	0	2
x_6	1	-1	1	1	0	0	1	4

Βήμα1:

Βρες το σύνολο $J_- = \{j : a_{0j} < 0\} = \{3\}$, επειδή $J_- \neq \varnothing$, ο αλγόριθμος δεν σταματάει.

Βήμα2:

Βρες το σύνολο $I_+ = \{i : a_{i0} > 0\} = \{1,3\}$, επειδή $I_+ \neq \varnothing$ το πρόβλημα δεν είναι απεριόριστο.

Βρες το δείκτη της εισερχόμενης μεταβλητής r = 3 διότι

$$\frac{b_r}{a_{r0}} = \min\left\{\frac{b_i}{a_{i0}} : i \in I_+\right\} = \min\left\{\frac{b_1}{a_{10}}, \frac{b_2}{a_{30}}\right\} = \min\{9/2, 4/1\} = 4$$

Βήμα3:

Θέσε $J_+ = \{j : a_{0j} > 0\} = \{1, 2\}$ και υπολόγισε

$$\theta_1 = \frac{-a_{0k}}{a_{rk}} = \min\left\{\frac{-a_{0j}}{arj} : j \in J_-, a_{rj} > 0\right\} = \min\left\{\frac{-a_{03}}{a_{33}}\right\} = \min\{-(-4)/1\} = 4$$

$$\theta_2 = \frac{-a_{0l}}{a_{rl}} = \min\left\{\frac{-a_{0j}}{arj} : j \in J_+, a_{rj} < 0\right\} = \min\left\{\frac{-a_{01}}{a_{31}}\right\} = \min\{-(1)/-1\} = 1$$

Επειδή $\theta_1 < \theta_2$ το s=1 και η εξερχόμενη μεταβλητή είναι η x_4

Βήμα4:

Σχημάτισε το επόμενο ταμπλό με στοιχείο περιστροφής το a_{31}

		x_1	x_2	x_3	x_4	x_5	x_6	Δεξί Μέρος
z	-3	0	2	-3	0	0	1	4
x_3	3	0	2	3	1	0	1	13
x_5	0	0	2	0	0	1	1	6
x_6	-1	1	-1	-1	0	0	-1	-4

Βήμα1:

Βρες το σύνολο $J_- = \{j : a_{0j} < 0\} = \{3\}$, επειδή $J_- \neq \varnothing$, ο αλγόριθμος δε σταματάει.

Βήμα2:

Βρες το σύνολο $I_+ = \{i : a_{i0} > 0\} = \{1\}$, επειδή $I_+ \neq \varnothing$ το πρόβλημα δεν είναι απεριόριστο.

Βρες το δείκτη της εισερχόμενης μεταβλητής r = 1 διότι

$$\frac{b_r}{a_{r0}} = \min\left\{\frac{b_i}{a_{i0}} : i \in I_+\right\} = \min\left\{\frac{b_1}{a_{10}}\right\} = \min\{13/3\} = 4.3$$

Βήμα3:

Θέσε $J_+ = \{j : a_{0j} > 0\} = \{2, 6\}$ και υπολόγισε

$$\theta_1 = \frac{-a_{0k}}{a_{rk}} = \min\left\{\frac{-a_{0j}}{arj} : j \in J_-, a_{rj} > 0\right\} = \min\left\{\frac{-a_{03}}{a_{13}}\right\} = \min\{-(-3)/3\} = 1$$

$$\theta_2 = \frac{-a_{0l}}{a_{rl}} = \min\left\{\frac{-a_{0j}}{arj} : j \in J_+, a_{rj} < 0\right\} = \min\{\varnothing\} = \infty$$

Επειδή $\theta_1 < \theta_2$ το s = 1 και η εξερχόμενη μεταβλητή είναι η x_6

Βήμα4:

Σχημάτισε το επόμενο ταμπλό με στοιχείο περιστροφής το a_{13}

		x_1	x_2	x_3	x_4	x_5	x_6	Δεξί Μέρος
z		0	4	0	1	0	2	17
x_3		0	2/3	1	1/3	0	1/3	13/3
x_5		0	2	0	0	1	1	6
x_1		1	-1/3	0	1/3	0	-2/3	1/3

Επειδή $J_- = \{j : a_{0j} < 0\} = \varnothing$, η τρέχουσα λύση είναι βέλτιστη και ισούται με $x_1 = 1/3$, $x_2 = 0$, $x_3 = 13/3$, $x_4 = 0$, $x_5 = 6$, $x_6 = 0$. Η αντικειμενική συνάρτηση ισούται με $z = 17$.

Παράδειγμα 2.6.2

Να λυθεί το παρακάτω γραμμικό πρόβλημα με τον αλγόριθμο Simplex εξωτερικών σημείων.

$$\max z = -2x_1 + 2x_2 - 3x_3 - 4x_4$$

$$\text{s.t.} \quad \begin{aligned} -x_1 - 2x_2 + 2x_3 - x_4 &\le 18 \\ -2x_1 - 12x_2 - x_3 - x_4 &\le 4 \\ 3x1 + x_2 - 2x_3 + 10x_4 &\le 12 \end{aligned}$$

$$x_1, x_2, x_3 \text{ και } x_4 \ge 0$$

Λύση:

Προσθέτουμε τις χαλαρές μεταβλητές x_5, x_6 και x_7, οπότε το γραμμικό πρόβλημα μετατρέπεται στο παρακάτω:

$$\text{μεγ. } z = -2x_1 + 2x_2 - 3x_3 - 4x_4$$

$$\text{μ. π. :} \quad \begin{aligned} -x_1 - 2x_2 + 2x_3 - x_4 + x_5 &= 18 \\ -2x_1 - 12x_2 - x_3 - x_4 + x_6 &= 4 \\ 3x1 + x_2 - 2x_3 + 10x_4 + x_7 &= 12 \end{aligned}$$

$$x_1, x_2, x_3, x_4, x_5, x_6 \text{ και } x_7 \ge 0$$

Βήμα0 (Αρχικοποίηση)

Από το αρχικό πρόβλημα κατασκευάζουμε το αρχικό ταμπλό του εξωτερικού simplex

		x_1	x_2	x_3	x_4	x_5	x_6	x_7	Δεξί Μέρος
z	-2	2	-2	3	4	0	0	0	0
x_3	-2	-1	-2	2	4	1	0	0	18
x_5	-12	-2	-12	-1	-1	0	1	0	4
x_1	1	3	1	-2	10	0	0	1	12

Βήμα1:

Βρες το σύνολο $J_- = \{j : a_{0j} < 0\} = \{2\}$, επειδή $J_- \neq \varnothing$, ο αλγόριθμος δε σταματά.

Βήμα2:

Βρες το σύνολο $I_+ = \{i : a_{i0} > 0\} = \{3\}$, επειδή $I_+ \neq \varnothing$ το πρόβλημα δεν είναι απεριόριστο.

Βρες το δείκτη της εισερχόμενης μεταβλητής r = 3 διότι

$$\frac{b_r}{a_{r0}} = \min\left\{\frac{b_i}{a_{i0}} : i \in I_+\right\} = \min\left\{\frac{b_3}{a_{30}}\right\} = \min\{12/1\} = 12$$

Βήμα3:

Θέσε $J_+ = \{j : a_{0j} > 0\} = \{1, 3, 4\}$ και υπολόγισε

$$\theta_1 = \frac{-a_{0k}}{a_{rk}} = \min\left\{\frac{-a_{0j}}{arj} : j \in J_-, a_{rj} > 0\right\} = \min\left\{\frac{-a_{02}}{a_{32}}\right\} = \min\{-(-2)/1\} = 2$$

$$\theta_2 = \frac{-a_{0l}}{a_{rl}} = \min\left\{\frac{-a_{0j}}{arj} : j \in J_+, a_{rj} < 0\right\} = \min\left\{\frac{-a_{03}}{a_{33}}\right\} = \min\{-(-3)/-2\} = 1.5$$

Επειδή $\theta_2 < \theta_1$ το s=3 και η εξερχόμενη μεταβλητή είναι η x_6

Βήμα4:

Κατασκεύασε το επόμενο simplex ταμπλό με το στοιχείο περιστροφής a_{33}

		x_1	x_2	x_3	x_4	x_5	x_6	x_7	Δεξί Μέρος
z	-1	13/2	-1/2	0	19	0	0	3/2	18
x_3	-25/2	2	-1	0	11	1	0	1	30
x_5	-12	-7/2	-25/2	0	-6	0	1	-1/2	-2
x_1	-1/2	-3/2	-1/2	1	-5	0	0	-1/2	-6

Βήμα1:

Βρες το σύνολο $J_- = \{j : a_{0j} < 0\} = \{2\}$, επειδή $J_- \neq \varnothing$, ο αλγόριθμος δε σταματάει.

Βήμα2:

Βρες το σύνολο $I_+ = \{i : a_{i0} > 0\} = \varnothing$, επειδή $I_+ = \varnothing$ το πρόβλημα είναι απεριόριστο.

2.7 Δυϊκός αλγόριθμος Simplex εξωτερικού σημείου

Θα περιγράψουμε τη δυϊκή μέθοδο Simplex εξωτερικού σημείου, η οποία επιλύει το δυϊκό πρόβλημα απευθείας στον (πρωτεύοντα) πίνακα Simplex. Έστω το τυπικό πρόβλημα γραμμικού προγραμματισμού:

$$\begin{aligned} &\text{Maximize} && z = cx \\ &\text{Subject to} && Ax = b \\ & && x \geq 0 \end{aligned} \qquad 2.7.1$$

Ξεκινάμε με ένα δυϊκό βασικό εφικτό σημείο και δημιουργούμε τον αντίστοιχο δυϊκό εξωτερικό πίνακα Simplex. Η επάνω γραμμή του πίνακα περιέχει στην αντικειμενική

συνάρτηση. Η τελευταία στήλη περιέχει τα στοιχεία της δεξιάς πλευράς. Στις υπόλοιπες θέσεις έχουμε τους συντελεστές του πίνακα Α. Ο αλγόριθμος ξεκινά με μία δυϊκή εφικτή βασική λύση. Εάν τα στοιχεία της δεξιάς πλευράς είναι μη αρνητικά, τότε ο αλγόριθμος σταματά. Σε αυτήν την περίπτωση η τρέχουσα BFS είναι βέλτιστη για το πρόβλημα 2.7.1. Αλλιώς, προσδιορίζουμε ένα σύνολο δεικτών $I_- = \{1 \le i \le m : b_i < 0\}$ και δημιουργούμε μία νέα γραμμή, η οποία προστίθεται στο κάτω μέρος του πίνακα. Τα στοιχεία της (m+1)-γραμμής δίνονται από τον τύπο $a_{m+1,j} = \sum_{i \in I_-} a_{ij}$ and $b_{m+1,n} = \sum_{i \in I_-} b_i$. Η (m+1)-γραμμή καλείται διακεκριμένη γραμμή ή γραμμή οδηγός.

Περιγραφή του Δυϊκού αλγορίθμου Simplex εξωτερικού σημείου

Βήμα0: **(Ξεκίνημα)**

Ξεκινάμε με ένα δυϊκό εφικτό βασικό σημείο και δημιουργούμε τον αντίστοιχο πίνακα του δυϊκού εξωτερικού αλγόριθμου Simplex.

Βήμα1: **(Έλεγχος βελτιστότητας)**

Προσδιορίζουμε το σύνολο των δεικτών $I_- = \{1 \le i \le m : b_i < 0\}$. Αν $I_- = \varnothing$, ΤΕΛΟΣ. Η τρέχουσα βασική λύση είναι βέλτιστη. Προσδιορίζουμε τη διακεκριμένη γραμμή από τον τύπο $a_{m+1,j} = \sum_{i \in I_-} a_{ij}$ and $b_{m+1,n} = \sum_{i \in I_-} b_i$

Βήμα2: **(Επιλογή της οδηγού στήλης)**

Προσδιορίζουμε το σύνολο των δεικτών $J_- = \{1 \le i \le n : a_{m+1,j} < 0\}$. Αν $J_- = \varnothing$, ΤΕΛΟΣ. Το πρόβλημα δεν είναι εφικτό.

Αλλιώς, προσδιορίζουμε τη θέση της εισαγομένης μεταβλητής εισόδου s από τη σχέση

$$\frac{a_{0s}}{a_{m+1,s}} = \min\left\{\frac{-a_{0j}}{a_{m+1,j}} : j \in J_-\right\}$$

Εισάγουμε τη μη βασική μεταβλητή x_s.

Βήμα3: **(Επιλογή της οδηγού γραμμής)**

Προσδιορίζουμε το σύνολο των δεικτών $I_+ = \{1 \le i \le m : i \notin I_-\}$ και υπολογίζουμε τα

$$\theta_1 = \frac{b_k}{a_{ks}} = \min\left\{\frac{b_i}{a_{is}} : i \in I_-, a_{is} < 0\right\}$$

$$\theta_2 = \frac{b_l}{a_{ls}} = \min\left\{\frac{b_i}{a_{is}} : i \in I_+, a_{is} > 0\right\}$$

Βρίσκουμε τη θέση της μεταβλητής r που θα βγάλουμε , αν $\theta_1 \le \theta_2$ θέσε r = k αλλιώς s = l και εξάγουμε τη βασική μεταβλητή $x_{B(r)}$.

Βήμα4: **(Περιστροφή)**

Σχηματίζουμε τον επόμενο πίνακα με τη βοήθεια της οδηγού μεταβλητής a_{rs} δηλ.

Θέτουμε $a_{rj} \leftarrow \frac{a_{rj}}{a_{rs}}$ όπου j=1, 2, … ,n, n+1

$$a_{ij} \leftarrow a_{ij} - \frac{a_{rj}}{a_{rs}} a_{is} \quad \text{όπου } i = 0, 1, 2, \ldots m \ (i \ne r) \text{ και } j = 1, 2, \ldots, n, n+1$$

Και πηγαίνουμε στο Βήμα1.

Παράδειγμα 2.7.1

Να λυθεί το επόμενο γραμμικό πρόβλημα με τον αλγόριθμο simplex εξωτερικού σημείου.

$$\begin{aligned} \max z = & -x_1 - 2x_2 - x_3 - x_4 \\ \text{s. t. :} \quad & -2x_1 + x_2 - 3x_3 - x_4 \le -4 \\ & x_1 + 2x_3 - x_4 \le -2 \\ & -x_1 + 2x_2 - 2x_3 - x_4 \le 15 \end{aligned}$$

$$2x_1 - x_2 + 2x_3 + x_4 \leq 1$$

$$x_1, x_2, x_3 \text{ και } x_4 \geq 0$$

Λύση:

Προσθέτουμε τις χαλαρές μεταβλητές x_5, x_6 , x_7 και x_8 έτσι ώστε το γραμμικό πρόβλημα να πάρει την πιο κάτω μορφή:

$$\begin{aligned}
\max z= \; & -x_1 - 2x_2 - x_3 - x_4 \\
\text{s. t. :} \; & -2x_1 + x_2 - 3x_3 - x_4 + x_5 = -4 \\
& x_1 + 2x_3 - x_4 + x_6 = -2 \\
& -x_1 + 2x_2 - 2x_3 - x_4 + x_7 = 15 \\
& 2x_1 - x_2 + 2x_3 + x_4 + x_8 = 1
\end{aligned}$$

$$x_1, x_2, x_3, x_4, x_5 x_6, x_7 \text{ και } x_8 \geq 0$$

Βήμα0: (Ξεκίνημα).

Από το αρχικό πρόβλημα, δημιουργούμε τον αρχικό δυϊκό εξωτερικό πίνακα simplex

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	R.H.S
z	1	2	1	1	0	0	0	0	0
x_5	-2	1	-3	-1	1	0	0	0	-4
x_6	1	0	2	-1	0	1	0	0	-2
x_7	-1	2	-2	-1	0	0	1	0	15
x_8	2	-1	2	1	0	0	0	1	1

Βήμα1: (Ελεγχος Βελτιστοποίησης)

Προσδιορίζουμε τους δείκτες $I_- = \{1 \leq i \leq m : b_i < 0\} = \{1, 2\}$, αφού $I_- \neq \varnothing$, δημιουργούμε τη διακεκριμένη γραμμή

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	R.H.S
z	1	2	1	1	0	0	0	0	0
x_5	-2	1	-3	-1	1	0	0	0	-4
x_6	1	0	2	-1	0	1	0	0	-2
x_7	-1	2	-2	-1	0	0	1	0	15
x_8	2	-1	2	1	0	0	0	1	1
	-1	1	-1	-2	1	1	0	0	-6

Βήμα2: (Επιλογή της οδηγού στήλης).

Προσδιορίζουμε τους δείκτες $J_- = \{1 \le i \le n : a_{m+1,j} < 0\} = (1, 2, 4\}$. Επιλέγουμε την οδηγό στήλη s

$$\frac{a_{0s}}{a_{m+1,s}} = \min\left\{\frac{-a_{0j}}{a_{m+1,j}} : j \in J_-\right\} = \min\left\{\frac{-a_{01}}{a_{5,1}}, \frac{-a_{02}}{a_{5,2}}, \frac{-a_{04}}{a_{5,4}}\right\} = \min\{-1/-1, -1/-1, -1/-2\} = 1/2$$

άρα s = 4 . Εισάγουμε το στοιχείο x_4.

Βήμα3: (Επιλογή της οδηγού γραμμής).

Προσδιορίζουμε το σύνολο δεικτών $I_+ = \{1 \le i \le m : i \notin I_-\} = \{3, 4\}$. Υπολογίζουμε τα θ_1 και θ_2

$$\theta_1 = \frac{b_k}{a_{ks}} = \min\left\{\frac{b_i}{a_{is}} : i \in I_-, a_{is} < 0\right\} = \min\left\{\frac{b_1}{a_{14}}, \frac{b_2}{a_{24}}\right\} = \min\{-4/-1, -2/-1\} = 2$$

$$\theta_2 = \frac{b_l}{a_{ls}} = \min\left\{\frac{b_i}{a_{is}} : i \in I_+, a_{is} > 0\right\} = \min\left\{\frac{b_4}{a_{44}}\right\} = \{1/1\} = 1.$$

Αφού $\theta_2 < \theta_1$ είναι r = 4, έτσι εξάγουμε την x_8.

Βήμα4: (Λειτουργία της Περιστροφής)

Στον πιο κάτω πίνακα με οδηγό στοιχείο το a_{44} έχουμε

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	R.H.S
z	-1	3	-1	0	0	0	0	-1	-1
x_5	0	0	-1	0	1	0	0	1	-3
x_6	3	-1	4	0	0	1	0	1	-1
x_7	1	1	0	0	0	0	1	1	16
x_4	2	-1	2	1	0	0	0	1	1

Επανάληψη 2

Βήμα1: (Ελεγχος βελτιστότητας).

Προσδιορίζουμε το σύνολο των δεικτών $I_- = \{1 \le i \le m : b_i < 0\} = \{1, 2\}$ αφού $I_- \neq \varnothing$ και στη συνέχεια δημιουργούμε τη διακεκριμένη γραμμή

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	R.H.S
z	-1	3	-1	0	0	0	0	-1	-1
x_5	0	0	-1	0	1	0	0	1	-3
x_6	3	-1	4	0	0	1	0	1	-1
x_7	1	1	0	0	0	0	1	1	16
x_4	2	-1	2	1	0	0	0	1	1
	3	-1	3	0	1	1	0	2	-4

Βήμα2: (Επιλογή της οδηγού στήλης).

Προσδιορίζουμε το σύνολο των δεικτών $J_- = \{1 \le i \le n : a_{m+1,j} < 0\} = (2\}$. Επιλέγουμε την οδηγό στήλη s

$$\frac{a_{0s}}{a_{m+1,s}} = \min\left\{\frac{-a_{0j}}{a_{m+1,j}} : j \in J_-\right\} = \min\left\{\frac{-a_{02}}{a_{5,2}}\right\} = \min\{-3/-1\} = 3$$

άρα s = 2 . Εισάγουμε την x_2.

Βήμα3: (Επιλογή της οδηγού γραμμής).

Προσδιορίζουμε το σύνολο των δεικτών $I_+ = \{1 \le i \le m : i \notin I_-\} = \{3, 4\}$. Υπολογίζουμε τα θ_1 και θ_2

$$\theta_1 = \frac{b_k}{a_{ks}} = \min\left\{\frac{b_i}{a_{is}} : i \in I_-, a_{is} < 0\right\} = \min\left\{\frac{b_2}{a_{24}}\right\} = \min\{-1/-1\} = 1.$$

$$\theta_2 = \frac{b_l}{a_{ls}} = \min\left\{\frac{b_i}{a_{is}} : i \in I_+, a_{is} > 0\right\} = \min\left\{\frac{b_3}{a_{32}}\right\} = \{16/1\} = 1.$$

Αφού $\theta_1 < \theta_2$ τότε r = 2 και έτσι εξάγουμε την x_6.

Βήμα4: (Λειτουργία της Περιστροφής).

Στον πιο κάτω πίνακα με οδηγό στοιχείο το a_{22} έχουμε

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	R.H.S
z	8	0	11	0	0	3	0	2	-4
x_5	0	0	-1	0	1	0	0	1	-3
x_2	-3	1	-4	0	0	-1	0	-1	1
x_7	4	0	4	0	0	1	1	22	15
x_4	-1	0	-2	1	0	-1	0	0	2

Επανάληψη 3

Βήμα1: (Έλεγχος Βελτιστότητας)

Προσδιορίζουμε το σύνολο των δεικτών $I_- = \{1 \le i \le m : b_i < 0\} = \{1\}$. Επομένως, δεν υπάρχει ανάγκη να δημιουργήσουμε τον εκτεταμένο πίνακα. Το ρόλο της διακεκριμένης γραμμής αναλαμβάνει η γραμμή 1.

Βήμα2: : (Επιλογή της οδηγού στήλης).

Προσδιορίζουμε το σύνολο των δεικτών $J_- = \{1 \le i \le n : a_{m+1,j} < 0\} = \{3\}$. Επιλέγουμε την οδηγό στήλη s

$$\frac{a_{0s}}{a_{m+1,s}} = \min\left\{\frac{-a_{0j}}{a_{m+1,j}} : j \in J_-\right\} = \min\left\{\frac{-a_{03}}{a_{1,3}}\right\} = \min\{-11/-1\} = 11$$

άρα s = 3 . Εισάγουμε την x_3.

Βήμα3: (Επιλογή της οδηγού γραμμής).

Προσδιορίζουμε το σύνολο των δεικτών $I_+ = \{1 \le i \le m : i \notin I_-\} = \{2, 3, 4\}$. Υπολογίζουμε τα θ_1 και θ_2

$$\theta_1 = \frac{b_k}{a_{ks}} = \min\left\{\frac{b_i}{a_{is}} : i \in I_-, a_{is} < 0\right\} = \min\left\{\frac{b_1}{a_{13}}\right\} = \min\{3/-1\} = -3.$$

$$\theta_2 = \frac{b_l}{a_{ls}} = \min\left\{\frac{b_i}{a_{is}} : i \in I_+, a_{is} > 0\right\} = \min\left\{\frac{b_3}{a_{33}}\right\} = \{15/4\} = 3.75.$$

Αφού $\theta_1 < \theta_2$ τότε r = 1 και έτσι εξάγουμε την x_5.

Βήμα4: (Λειτουργία της Περιστροφής).

Στον πιο κάτω πίνακα με οδηγό στοιχείο το a_{13} έχουμε

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	R.H.S
z	8	0	0	0	-11	3	0	13	-37

x_3	0	0	1	0	-1	0	0	-1	3
x_2	-3	1	0	0	-4	-1	0	-5	13
x_7	4	0	0	0	4	1	1	6	3
x_4	-17	0	0	1	-2	-1	0	-2	8

Επανάληψη 4

Βήμα1: (Έλεγχος Βελτιστότητας)

Προσδιορίζουμε το σύνολο των δεικτών $I_- = \{1 \le i \le m : b_i < 0\} = \varnothing$ αφού $I_- = \varnothing$ τότε το πρόβλημα είναι βέλτιστο για $x_1 = 0$, $x_2 = 13$,$x_3 = 3$,$x_4 = 8$, $x_5 = x_6 = x7 = x_8 = 0$ και $z = -37$.

ΚΕΦΑΛΑΙΟ 3

ΑΛΓΟΡΙΘΜΟΙ ΜΟΡΦΗΣ ΑΝΑΘΕΩΡΗΜΕΝΟΥ SIMPLEX

3.1 Εισαγωγή

Η μέθοδος Simplex, η πιο γνωστή μέθοδος για την επίλυση προβλημάτων Γραμμικού Προγραμματισμού (LPs), έχει δύο βασικές παραλλαγές. Πρόκειται για την αναθεωρημένη μέθοδο Simplex και την τυποποιημένη ή πλήρους πίνακα μέθοδο Simplex. Στις μέρες μας, πρακτικά όλες οι σοβαρές υλοποιήσεις ανήκουν στην κατηγορία των αναθεωρημένων μεθόδων Simplex, επειδή είναι πιο αποτελεσματικές για αραιά γραμμικά προβλήματα (sparse LPs) που είναι τα πιο συνηθισμένα. Η αναθεωρημένη μέθοδος Simplex είναι μία συστηματική διαδικασία για την υλοποίηση των βημάτων της μεθόδου Simplex σε μικρότερο πίνακα, κάνοντας έτσι εξοικονόμηση στον απαιτούμενο αποθηκευτικό χώρο.

Στο κεφάλαιο αυτό παρουσιάζουμε αλγόριθμους simplex και simplex εξωτερικού σημείου στην αναθεωρημένη μορφή. Επειδή οι κανόνες περιστροφής περιγράφουν τον τρόπο με τον οποίο εισάγουμε και εξάγουμε μεταβλητές, θα παρουσιάσουμε 8 διαφορετικούς κανόνες περιστροφής.

Αφού σε κάθε επανάληψη στους αλγόριθμους τύπου simplex, η συνολική εργασία επικεντρώνεται στον προσδιορισμό του αντιστρόφου της βάσης, θα παρουσιάσουμε δύο διαφορετικούς τρόπους για την ανανέωση του αντιστρόφου

της βάσης. Οι πιο γνωστές τεχνικές είναι (i) η μορφή γινομένου του αντιστρόφου (PFI), (ii) η μορφή εξάλειψης του αντιστρόφου (EFI), και (iii) μία πρόσφατα αναπτυγμένη τεχνική από τον Benhamadou 2002 .

3.2 Η άλγεβρα του αλγόριθμου Simplex

Έστω το τυποποιημένο πρόβλημα γραμμικού προγραμματισμού:

$$\begin{aligned} &\text{Maximize} && z = cx \\ &\text{Subject to:} && Ax = b \\ & && x \geq 0 \end{aligned} \qquad 3.2.1$$

όπου $A \in R^{m+n}$, $c,x \in R^{n}$, $b \in R^{m}$.

Η βασική εφικτή λύση στο πρόβλημα αυτό αντιστοιχεί σε ένα ακραίο σημείο της εφικτής περιοχής και μαθηματικά προσδιορίζεται με τον χωρισμό του πίνακα A σε έναν αντιστρέψιμο πίνακα βάσης B και τον πίνακα των μη-βασικών στηλών N. Δηλαδή,

$$A = (B: N) \qquad 3.2.2$$

Βασιζόμενοι σε αυτόν τον χωρισμό, το γραμμικό σύστημα Ax = b μπορεί να ξαναγραφτεί δίνοντας

$$Bx_B + Nx_N = b \qquad 3.2.3$$

Αυτό απλοποιείται σε

$$x_B + B^{-1}Nx_N = B^{-1}b \qquad 3.2.4$$

και λύνοντας ως προς x_B με τη βοήθεια του x_N δίνει

$$x_B = B^{-1}b - B^{-1}Nx_N \qquad 3.2.5$$

Θέτοντας τώρα $x_N = 0$, βλέπουμε ότι η 2.2.5 δίνει $x_B = B^{-1}b$. Η λύση

$$x=\begin{pmatrix} x_B \\ x_N \end{pmatrix}=\begin{pmatrix} B^{-1}b \\ 0 \end{pmatrix}$$

καλείται βασική λύση, το διάνυσμα x_B καλείται διάνυσμα των βασικών μεταβλητών, και το x_N καλείται διάνυσμα των μη-βασικών μεταβλητών. Εάν επιπρόσθετα, $x_B = B^{-1} b \geq 0$, τότε

$$x=\begin{pmatrix} B^{-1}b \\ 0 \end{pmatrix}$$

και καλείται βασική εφικτή λύση.

Θεωρείστε τώρα την αντικειμενική συνάρτηση z = cx. Χωρίστε το διάνυσμα κόστους c σε βασικά και μη-βασικά στοιχεία δηλ. $c = (c_B, c_N)$. Η αντικειμενική συνάρτηση γίνεται

$$z=c_B x_B+c_N x_N \qquad 3.2.6$$

Αντικαθιστώντας τώρα την έκφραση του x_B που ορίζεται στη 2.2.6 στην 4.2.5 παίρνουμε

$$z=c_B(B^{-1}b-B^{-1}Nx_N)+c_N x_N \qquad 3.2.7$$

Το οποίο μπορεί να γραφεί σαν $z=c_B B^{-1}b-(c_B B^{-1}N-c_N)x_N$ 3.2.8

Έχουμε εκφράσει έτσι το z σαν τη σταθερά $c_B B^{-1}b$ μείον τον όρο $(c_B B^{-1}N-c_N)x_N$. Θέτοντας $x_N = 0$, βλέπουμε ότι η 2.2.8 δίνει $z=c_B B^{-1}b$, που είναι η αντικειμενική τιμή που αντιστοιχεί στην τρέχουσα βασική εφικτή λύση. Επομένως, το τρέχον ακραίο σημείο λύσης μπορεί να αναπαρασταθεί σε κανονική μορφή:

$$z=c_B B^{-1}b-(c_B B^{-1}N-c_N)x_N \qquad 3.2.9$$

$$x_B = B^{-1}b - B^{-1}Nx_N \qquad 3.2.10$$

με την τρέχουσα βασική εφικτή λύση να δίνεται από τη σχέση

$$z=c_B B^{-1}b \qquad 3.2.11$$

$$x=\begin{pmatrix} x_B \\ x_N \end{pmatrix}=\begin{pmatrix} B^{-1}b \\ 0 \end{pmatrix}\geq 0 \qquad 3.2.12$$

Αναδιατάσσοντας τώρα τους όρους έτσι ώστε όλες οι αναθεωρημένες μεταβλητές να βρίσκονται στο αριστερό μέρος της εξίσωσης και τις σταθερές στη δεξιό μέρος, έχουμε

$$z+(c_B B^{-1}N-c_N)x_N=c_B B^{-1}b \qquad 3.2.13$$

$$x_B + B^{-1}Nx_N = B^{-1}b \qquad 3.2.14$$

3.3 Ο αναθεωρημένος αλγόριθμος Simplex

Βήμα0: **(Ξεκίνημα)**

Ξεκινάμε με μία βασική εφικτή βάση B. Θέστε $N=\{1,2,\ldots,n\}\setminus B$ και υπολογίστε τους πίνακες B, N και τον αντίστροφο πίνακα B^{-1}.

Θέστε $x_B = B^{-1}b$, $w^T = (c_B)^T B^{-1}$ και $s_j = w^T a_j - c_j$ (j=1, 2, …, n)

Βήμα1: **(Έλεγχος Βελτιστότητας)**

Θέστε $J=\{j: j\in N \text{ and } s_j < 0\}$. Αν J = ∅, ΤΕΛΟΣ. Η λύση είναι βέλτιστη. Αλλιώς, με βάση έναν συγκεκριμένο κανόνα περιστροφής, επιλέξτε την εισερχόμενη μεταβλητή με βάση το δείκτη $l=N[t]\in J$. Η μεταβλητή x_l είναι η εισερχόμενη μεταβλητή.

Βήμα2: **(Επιλογή του Ελάχιστου Λόγου)**

Θέστε $h_l = B^{-1}a_l$ $I_+ = \{ i : 1\leq i \leq m \text{ και } h_{il} > 0\}$. Αν $I_+ = \varnothing$, ΤΕΛΟΣ. Το πρόβλημα δεν είναι φραγμένο. Αλλιώς βρείτε τον δείκτη της οδηγού στήλης r από τη σχέση:

$$\frac{x_{B[r]}}{h_{rl}} = \min\left\{\frac{x_{B[i]}}{h_{il}}: i\in I_+\right\}$$

Επομένως η μεταβλητή $x_{B[r]} = x_k$ είναι η εξερχόμενη μεταβλητή.

Βήμα3: **(Περιστροφή)**

Θέστε N[t] = k και B[r] = l. Περιστρέψτε με βάση την a_{rs} με τον συνήθη τρόπο. Θα ενημερωθούν έτσι τα B^{-1} , w, x_B, s. Πηγαίνετε στο Βήμα1.

3.4 Περιγραφή του αναθεωρημένου αλγόριθμου simplex

Στην παράγραφο αυτή περιγράφουμε τον αναθεωρημένο αλγόριθμο simplex βήμα προς βήμα.

Βήμα0: **(Ξεκίνημα)**

Ξεκινάμε με μία βασική εφικτή βάση B. Για να βρούμε το w λύνουμε το γραμμικό σύστημα

$$w^T B = c_B^T$$

Πριν την βελτιστοποίηση τα στοιχεία του w αναφέρονται σαν σκιώδεις τιμές. Όταν επέλθει βελτιστοποίηση αποτελούν μία εφικτή δυϊκή λύση. Υπολογίζουμε τα ελαττωμένα κόστη των μη βασικών μεταβλητών: $s_j = w^T a_j - c_j$ (j=1, 2, …, n)

Βήμα1: **(Έλεγχος Βελτιστότητας)**

Θέτουμε $J = \{j: j \in N \text{ and } s_j < 0\}$. Αν $J = \varnothing$, ΤΕΛΟΣ. Η λύση είναι βέλτιστη. Αλλιώς επιλέγουμε το δείκτη της εισερχόμενης μεταβλητής $l \in J$.

Αυτή είναι η φάση της βελτιστοποίησης που μας δίνει την ελαστικότητα να μεταβάλλουμε το κριτήριο που χρησιμοποιούμε ώστε να επιλέξουμε την εισερχόμενη μεταβλητή. Η επιλογή μίας "καλής" εισερχόμενης μεταβλητής είναι ένα από τα βασικά ερωτήματα που θα προσπαθήσουμε να απαντήσουμε.

Βήμα2: **(Επιλογή του Ελαχίστου Λόγου)**

Χρειαζόμαστε τώρα μία αναπαράσταση της νέας εισερχόμενης μεταβλητής ως προς την τρέχουσα βάση. Έστω a_l η στήλη του A_N (οι στήλες του πίνακα περιορισμών A που αντιστοιχούν στο μη-βασικό N) που σχετίζεται με την εισερχόμενη μεταβλητή και επιλύουμε το γραμμικό σύστημα $h_l = B^{-1}a_l$

Θέτουμε $I_+ = \{ i : 1 \le i \le m \text{ and } h_{il} > 0 \}$. Αν $I_+ = \varnothing$, ΤΕΛΟΣ. Το πρόβλημα δεν είναι φραγμένο. Αλλιώς βρίσκουμε το δείκτη της οδηγού στήλης r από τη σχέση:

$$\frac{x_{B[r]}}{h_{rl}} = \min\left\{\frac{x_{B[i]}}{h_{il}} : i \in I_+\right\}$$

Έτσι η μεταβλητή $x_{B[r]} = x_k$ είναι η εξερχόμενη μεταβλητή.

Βήμα3: **(Περιστροφή)**

Θέτουμε N[t] = k και B[r] = l. Περιστρέφουμε με βάση το a_{rs} με το συνηθισμένο τρόπο. Ενημερώνουμε έτσι τα B^{-1} , w, x_B, s.

Μπορούμε να συνοψίσουμε τη διαδικασία μίας απλής επανάληψης της αναθεωρημένης μεθόδου simplex ως ακολούθως:

- Επιλύστε το m x m γραμμικό σύστημα

$$\boxed{w^T B = c_B^T} \qquad 3.4.1.$$

- Υπολογίστε τα ελαττωμένα κόστη (n – m εσωτερικά γινόμενα)

$$\boxed{s_j = w^T a_j - c_j} \quad (j=1, 2, \ldots, n) \qquad 3.4.2.$$

- Επιλύστε το m x m γραμμικό σύστημα

$$\boxed{h_l = B^{-1}a_l} \qquad 3.4.3.$$

- Εφαρμόστε τον έλεγχο λόγου

$$\frac{x_{B[r]}}{h_{rl}} = \min\left\{\frac{x_{B[i]}}{h_{il}}: \; i \in I_{+}\right\} \qquad 3.4.4.$$

Εάν n >> m, ο υπολογισμός των ελαττωμένων κοστών της εξίσωσης 3.4.2. αποτελεί το επικρατούν κόστος του υπολογισμού.

3.5 Κανόνες περιστροφής

Ο υπολογισμός των ελαττωμένων κοστών στο βήμα 0 του αναθεωρημένου αλγορίθμου simplex αναφέρεται σαν "pricing out" των μη-βασικών στηλών. Ο στόχος του υπολογισμού των ελαττωμένων κοστών s_j είναι να βρούμε μία μη-βασική στήλη που θα ελαττώσει την τιμή της αντικειμενικής συνάρτησης εάν την εισάγουμε στη βάση. Αφού οποιαδήποτε στήλη με αρνητικό ελαττωμένο κόστος ικανοποιεί αυτό το κριτήριο, υπάρχουν διάφοροι εναλλακτικοί τρόποι να επιτύχουμε τον στόχο μας. Σε αυτή την παράγραφο περιγράφουμε σύντομα μερικούς από τους κανόνες περιστροφής. Περισσότερες λεπτομέρειες μπορείτε να βρείτε στις αναφορές.

3.5.1 Ο κανόνας του Dantizg

Ο πρώτος κανόνας περιστροφής έχει το όνομα του ανθρώπου που τον εφηύρε, του G. Dantzig. Ο κανόνας του Dantizg είναι επίσης γνωστός σαν ο κανόνας του μέγιστου συντελεστή. Είναι εύκολος στην υλοποίηση και υπολογίζει μία βέλτιστη λύση σύντομα. Παρόλο που είναι παλιός, θεωρείται πολύ αποτελεσματικός, κυρίως για προβλήματα της πραγματικής ζωής.

Η πολυπλοκότητα του κανόνα του Dantizg είναι εκθετική στη χειρότερη περίπτωση. Είναι τετριμμένο να κατασκευαστούν προβλήματα που επιδεικνύουν τέτοια συμπεριφορά (Keely , Minty 19972).

3.5.2 Ο κανόνας του Bland

Ο Bland αναθεώρησε έναν πολύ απλό κανόνα περιστροφής. Απέδειξε ότι η μέθοδος του αποφεύγει την κύκλωση (μία μη επιθυμητή κατάσταση κατά την οποία ο αλγόριθμος επαναλαμβάνει τις ίδιες λύσεις) και τερματίζει πάντοτε μετά από πεπερασμένο πλήθος επαναλήψεων. Παρόλα αυτά ο κανόνας του Bland, σε πραγματικά προβλήματα, παρουσιάζει ένα φαινόμενο που ονομάζεται παγίωση (*stalling)* (Bland 1977). Αυτό σημαίνει ότι ο αλγόριθμος δεν μπορεί να προχωρήσει σε καλύτερη λύση σε κάθε επανάληψη. Εξαιτίας αυτού, ο κανόνας του Bland συνήθως χρειάζεται περισσότερες επαναλήψεις από άλλους κανόνες και δεν θεωρείται πολύ καλή λύση.

3.5.3 Ο κανόνας της στοίβας

Ο κανόνας της στοίβας χρησιμοποιεί δύο στοίβες, μία για τις βασικές μεταβλητές και μία για τις μη-βασικές μεταβλητές. Παίρνουμε τις επιλεγόμενες μεταβλητές από την κορυφή της μίας στοίβας και τις εισάγουμε στην κορυφή της άλλης σύμφωνα με τη βασική πειθαρχία LIFO (Last In First Out). Ο κανόνας της στοίβας, όπως και ο κανόνας του Bland, δεν παρουσιάζει το πρόβλημα της κύκλωσης. Σε πραγματικά προβλήματα είναι πιο αποτελεσματικός από τον κανόνα του Bland.

3.5.4 Ο κανόνας της Ουράς

Ο κανόνας της ουράς λειτουργεί όπως ο κανόνας του σωρού. Η διαφορά είναι ότι οι δύο δομές λειτουργούν σύμφωνα με την πειθαρχία FIFO (First In First Out).

3.5.5 Η μέθοδος της πιο πρόσφατης θεώρησης (Least Recently Considered)

Η μέθοδος της πιο πρόσφατης θεώρησης (LRC) παρουσιάστηκε από τον (Zadeh 1980). Μπορεί να υλοποιηθεί εύκολα και δεν παρουσιάζει το φαινόμενο του stalling [10]. Η πολυπλοκότητα χειρότερης περίπτωσης της μεθόδου LRC θεωρείται πολυωνυμική αλλά δεν έχει δοθεί ακόμη απόδειξη (Gartner 1995), αλλά ούτε και έχουν βρεθεί προβλήματα στα οποία η μέθοδος LRC απαιτεί εκθετικό χρόνο.

3.5.6 Ο κανόνας μέγιστης αύξησης

Σύμφωνα με τον κανόνα αυτόν, η εισερχόμενη μεταβλητή θα φέρει την μεγαλύτερη βελτίωση σε σχέση με κάθε άλλη αποδεκτή μεταβλητή. Αυτή η μέθοδος περιγράφτηκε πρώτα στο (Keely, Minty 1972) . Μπορεί να βρει τη βέλτιστη λύση σε σχετικά λίγες επαναλήψεις γιατί προσπαθεί να εκμεταλλευτεί όλα τα δεδομένα. Το μειονέκτημά της είναι οι μεγάλες υπολογιστικές ανάγκες που παρουσιάζει σε κάθε επανάληψη. Επιπλέον, η πολυπλοκότητα χειρότερης περίπτωσης για τη μέθοδο αυτή είναι εκθετική σύμφωνα με τον (Gartner 1995).

3.5.7 Ο κανόνας της μερικής τιμολόγησης (partial pricing rule)

Η βασική ιδέα στον κανόνα αυτόν είναι να διαιρέσουμε το σύνολο των μη βασικών μεταβλητών και να εξετάσουμε ένα υποσύνολο κάθε φορά για τις αποδεκτές μεταβλητές. Το κέρδος μας είναι λιγότεροι υπολογισμοί σε κάθε επανάληψη. Τα υποσύνολα μπορούν να σχηματιστούν δυναμικά κατά τη διάρκεια των επαναλήψεων είτε στατικά πριν από την εκκίνηση του βασικού βρόχου του αλγόριθμου.

3.5.8 Κανόνας μεγαλύτερης μεταβολής (steepest edge rule)

Ο κανόνας αυτός βασίζεται στη γεωμετρία του γραμμικού προβλήματος. Στην απλή του μορφή οδηγεί τον αλγόριθμο simplex στη βέλτιστη λύση σε λιγότερες επαναλήψεις από κάθε άλλη μέθοδο, αλλά έχει τις μεγαλύτερες υπολογιστικές απαιτήσεις ανα επανάληψη. Η απόδοσή του μπορεί να βελτιωθεί με χρήση προσεγγιστικών μεθόδων όπως περιγράφονται από τους (Swietanowski 1998) και (Vanderbei 2001). Η μέθοδος είναι επίσης κατάλληλη για παράλληλες υλοποιήσεις (Thomadakis et al 1996).

3.6 Ενημέρωση του αντιστρόφου της βάσης

Η κάθε επανάληψη σε αλγορίθμους τύπου simplex είναι σχετικά χρονοβόρα. Το πιο απαιτητικό κομμάτι της κάθε επανάληψης είναι ο προσδιορισμός του αντιστρόφου της βάσης, B^{-1}. Αυτός ο αντίστροφος, όμως, δεν είναι απαραίτητο να υπολογίζεται από την αρχή σε κάθε επανάληψη. Οι αλγόριθμοι τύπου Simplex διατηρούν ένα μέρος της βάσης B και κάνουν ενημέρωση της βάσης σε κάθε επανάληψη. Υπάρχουν διάφοροι τρόποι ενημέρωσης του αντιστρόφου της βάσης. Οι πιο γνωστές μέθοδοι είναι (i) η

μορφή γινομένου του αντιστρόφου (PFI), (ii) η μορφή εξάλειψης του αντιστρόφου (EFI), και (iii) μία πρόσφατα αναπτυγμένη τεχνική από τον Benhamadou 2002.

3.6.1 Ανάλυση των μεθόδων ενημέρωσης

Ο πολλαπλασιασμός πινάκων είναι μία βασική λειτουργία της γραμμικής άλγεβας η οποία έχει πολυάριθμες εφαρμογές στη θεωρία και στην εφαρμογή της Πληροφορικής. Πολλές εφαρμογές οφείλονται στο γεγονός ότι ο πολλαπλασιασμός πινάκων είναι ένα σημαντικό κομμάτι πολλών επιτυχημένων αλγορίθμων για υπολογιστικά προβλήματα της Επιχειρησιακής Έρευνας, όπως π.χ. κατά τη λύση γραμμικών εξισώσεων, την αντιστροφή πίνακα κτλ. Ο υπολογιστικός χρόνος που απαιτείται για τον πολλαπλασιασμό πινάκων αποτελεί το σημαντικότερο μέρος του συνολικού χρόνου που απαιτείται για την επίλυση τέτοιων προβλημάτων.

Πρόταση 3.6.1.1 Ο πολλαπλασιασμός δύο πυκνών (dense) τετραγωνικών πινάκων mxm απαιτεί m^3 πολλαπλασιασμούς και $m^3 - m^2$ προσθέσεις. Επομένως, η συνολική πολυπλοκότητα είναι $\Theta(m^3)$.

Πρόταση 3.6.1.2. Δύο βάσεις $(A_B)_{[1]}$, $(A_B)_{[2]} \in \Re^m$ ονομάζονται γειτονικές εάν διαφέρουν μεταξύ τους μόνο σε μία στήλη.

Σε κάθε επανάληψη, οι αλγόριθμοι τύπου simplex κινούνται από μία βάση σε μία γειτονική μέχρι να τερματίσουν. Οι λειτουργίες των επαναλήψεων τύπου simplex προσδιορίζονται με όρους αντιστροφής της τρέχουσας βάσης. Ο προσδιορισμός του $(A_B)^{-1}$ απαιτεί σημαντική υπολογιστική προσπάθεια. Εάν έπρεπε να υπολογίζεται σε κάθε επανάληψη θα ήταν υπολογιστικά απαγορευτικό.

3.6.2. Μορφή γινομένου του αντιστρόφου (Product Form of the Inverse)

Η βασική ιδέα είναι η εξής: Η τρέχουσα αντίστροφη βάση $(\tilde{B})^{-1}$ μπορεί να υπολογιστεί από την προηγούμενη αντίστροφη B^{-1} με μία απλή λειτουργία περιστροφής. Δηλαδή, έχουμε

$$(\tilde{B})^{-1} = E^{-1}B^{-1} \qquad (3.6.2.1)$$

όπου E^{-1} είναι ο αντίστροφος του πίνακα ήτα (eta-matrix). Χρησιμοποιούμε τα πιο κάτω βήματα για τον υπολογισμό του $(\tilde{B})^{-1}$.

Έστω ότι l και r είναι οι δείκτες της εισερχόμενης και της εξερχόμενης μεταβλητής αντίστοιχα.

Υπολογίζουμε την οδηγό στήλη $h_l = B^{-1} a_l$

Υπολογίζουμε το διάνυσμα στήλη

$$v = \left[-\frac{h_{1l}}{h_{rl}} \quad \cdots \quad \frac{1}{h_{rl}} \quad \cdots \quad -\frac{h_{ml}}{h_{rl}} \right]^T$$

Αντικαθιστούμε την *r* στήλη του ταυτοτικού πίνακα με το διάνυσμα στήλη v. Αυτός είναι ο αντίστροφος πίνακας του πίνακα ήτα (eta-matrix).

Υπολογίζουμε το νέο αντίστροφο βάσης με χρήση της σχέσης (3.6.2.1).

Ο αντίστροφος του πίνακα ήτα (eta-matrix) έχει την πιο κάτω μορφή

$$E^{-1} = I - \frac{1}{h_{rl}}(h_l - e_l)e_l^T = \begin{bmatrix} 1 & & -h_{1l}/h_{rl} & & \\ & \ddots & \vdots & & \\ & & 1/h_{rl} & & \\ & & \vdots & \ddots & \\ & & -h_{ml}/h_{rl} & & 1 \end{bmatrix}$$

Εάν ο τρέχων αντίστροφος βάσης έχει υπολογιστεί χρησιμοποιώντας τον κανονικό πολλαπλασιασμό στη σχέση (3.6.2.1), τότε η πολυπλοκότητα της μεθόδου (PFI) είναι $\Theta(m^3)$.

Μπορούμε να παρατηρήσουμε ότι ο πίνακας E^{-1} έχει ειδική μορφή. Συγκεκριμένα, ο E^{-1} είναι αραιός (sparse) και σε κάθε επανάληψη περιέχει $2m - 1$ μη μηδενικά στοιχεία. Για να μετρήσουμε το συνολικό αριθμό πολλαπλασιασμών και προσθέσεων που χρειάζονται για τη μέθοδο ανανέωσης, κάνουμε τις ακόλουθες υποθέσεις:

(i) Μετράμε μόνο τους πολλαπλασιασμούς ανάμεσα σε μη μηδενικούς αριθμούς αβ, όπου

($\alpha \in \Re^* - \{1\}$ and $\beta \in \Re^*$) ή ($\alpha \in \Re^*$ and $\beta \in \Re^* - \{1\}$) και

(ii) Μετράμε μόνο τις προσθέσεις μεταξύ μη μηδενικών αριθμών.

Η πιο πάνω μέθοδος ενημέρωσης του αντιστρόφου απλοποιείται στο σχήμα 3.6.2.1.

$$\underset{E^{-1}}{\begin{bmatrix} 1 & & -h_{1l}/h_{rl} & & \\ & \ddots & \vdots & & \\ & & 1/h_{rl} & & \\ & & \vdots & \ddots & \\ & & -h_{ml}/h_{rl} & & 1 \end{bmatrix}} \quad \underset{(A_B)^{-1}}{\begin{bmatrix} x & & x & & x \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x & & x & & x \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x & & x & & x \end{bmatrix}}$$

Σχήμα 3.6.2.1: Η μέθοδος ενημέρωσης PFI

Κατά την *m* επανάληψη, ο αντίστροφος βάσης έχει m^2 μη μηδενικά στοιχεία. Η r γραμμή του E^{-1} έχει μόνο ένα μη μηδενικό στοιχείο. Όλες οι άλλες (m-1) γραμμές

έχουν 2 μη μηδενικά στοιχεία. Για να υπολογίζουμε το γινόμενο στο σχήμα 3.6.2.1, σύμφωνα με τις πιο πάνω υποθέσεις, χρειαζόμαστε

(m–1) γραμμές x m πολλαπλασιασμοί + 1 x m πολλαπλασιασμοί = m^2 πολλαπλασιασμοί

και (m – 1) γραμμές x m προσθέσεις = m^2 – m προσθέσεις.

Το συνολικό κόστος της πιο πάνω μεθόδου είναι $2m^2$-m πράξεις (πολλαπλασιασμοί και προσθέσεις).

Μπορούμε τώρα να πούμε ότι αν χρησιμοποιήσουμε την πιο πάνω ιδιότητα του ήτα πίνακα (eta-matrix), η πολυπλοκότητα της σχέσης 3.6.2.1 είναι $\Theta(m^2)$ αλλιώς η πολυπλοκότητα είναι $\Theta(m^3)$.

3.6.3 Μία τροποποίηση της μεθόδου PFI

Η μέθοδος αυτή ανανέωσης παρουσιάστηκε από τον Benhamadou 2002. Η κύρια ιδέα είναι η ακόλουθη: Η τρέχουσα αντίστροφη βάση $(\tilde{B})^{-1}$ μπορεί να υπολογιστεί από την προηγούμενη αντίστροφη B^{-1} με ένα απλό εξωτερικό γινόμενο δύο διανυσμάτων και μία πρόσθεση πινάκων. Συγκεκριμένα έχουμε:

$$(\tilde{B})^{-1} = (\bar{B})^{-1}_{r.} + v \otimes (B)^{-1}_{r.} \qquad (3.6.3.1)$$

Όπου $(\bar{B})^{-1}_{r.}$ είναι ο πίνακας B^{-1} κάνοντας την *r* γραμμή μηδέν και $(B)^{-1}_{r.}$ είναι η *r* γραμμή του αντιστρόφου της βάσης. Περιγραφή της μεθόδου δίνεται πιο κάτω:

Έστω ότι l και r είναι οι δείκτες της εισερχόμενης και εξερχόμενης μεταβλητής, αντίστοιχα.

Υπολογίζουμε την οδηγό στήλη $h_l = B^{-1}a_l$

Υπολογίζουμε το διάνυσμα

$$v = \left[-\frac{h_{1l}}{h_{rl}} \quad \cdots \quad \frac{1}{h_{rl}} \quad \cdots \quad -\frac{h_{ml}}{h_{rl}} \right]^T$$

Υπολογίζουμε το εξωτερικό γινόμενο $v \otimes (B)^{-1}_{r.}$

Θέτουμε την *r* γραμμή του B^{-1} ίση με μηδέν. Αποθηκεύουμε τα αποτελέσματα στο $(\bar{B})^{-1}_{r.}$

Υπολογίζουμε το νέο αντίστροφο βάσης από τη σχέση (3.6.3.1).

Η πιο πάνω μέθοδος ανανέωσης απλοποιείται στο σχήμα 3.6.3.1.

$$B^{-1}_{r.}: \begin{bmatrix} b_{r1} & \cdots & b_{rr} & \cdots & b_{rm} \end{bmatrix}$$

$$(\tilde{B})^{-1} = \begin{bmatrix} b_{11} & \cdots & b_{1m} \\ \cdots & \cdots & \cdots \\ 0 & 0 & 0 \\ \cdots & \cdots & \cdots \\ b_{m1} & \cdots & b_{mm} \end{bmatrix} + \begin{bmatrix} -\frac{h_{1l}}{h_{rl}} \\ \vdots \\ \frac{1}{h_{rl}} \\ \vdots \\ -\frac{h_{ml}}{h_{rl}} \end{bmatrix} \quad [mxm]$$

y

Σχήμα 3.6.3.1: Μία τροποποίηση της μεθόδου (PFI)

Το εξωτερικό γινόμενο απαιτεί m^2 πολλαπλασιασμούς και η πρόσθεση των δύο πινάκων απαιτεί m^2 προσθέσεις. Το συνολικό κόστος της πιο πάνω μεθόδου είναι $2m^2$ πράξεις (πολλαπλασιασμοί και προσθέσεις). Επομένως η πολυπλοκότητα είναι $\Theta(m^2)$.

Η τροποποιημένη αυτή μέθοδος (MPFI) είναι πιο γρήγορη από την (PFI) όπως θα δούμε πιο κάτω.

3.6.4 Αλγόριθμος Simplex εξωτερικού σημείου

Ο πρώτος πολυωνυμικός αλγόριθμος παρουσιάστηκε από τον Khachian το 1979 και καλείται ελλειψοειδής αλγόριθμος. Στην πράξη η απόδοση του αλγορίθμου αυτού δεν είναι καλή για πραγματικά προβλήματα σε σχέση με τη μέθοδο simplex. Το 1984, ο Karmarkar παρουσίασε έναν νέο πολυωνυμικό αλγόριθμο για το γραμμικό πρόβλημα (LP) ο οποίος προσεγγίζει τη βέλτιστη λύση από το εσωτερικό της εφικτής περιοχής (Kamarkar 1984). Από τότε που δημοσιεύτηκε η εργασία του Karmarkar, πολλές εργασίες έχουν γραφτεί για τις μεθόδους εσωτερικών σημείων (Roos et al 1997). Όπως δείχνουν οι θεωρητικές τους ιδιότητες, οι μέθοδοι εσωτερικών σημείων λειτουργούν ικανοποιητικά για προβλήματα της πραγματικής ζωής και έχουν καλύτερη απόδοση από τη μέθοδο simplex για προβλήματα πολύ μεγάλης διάστασης (Lustig et al 1994).

Η μέθοδος EPSA αρχικά αναπτύχθηκε από τον κ. Παππαρίζο για το πρόβλημα assignment (Paparrizos 1991). Αργότερα, ο κ. Παπαρρίζος γενίκευσε τη μέθοδό εξωτερικών σημείων στο γενικευμένο γραμμικό πρόβλημα αναπτύσσοντας έναν δυϊκό, στη φύση του, αλγόριθμο (Paparrizos 1990). Ανεξάρτητα, οι (Anstreicher and Terlaky 1994), ανέπτυξαν έναν παρόμοιο πρωτεύοντα αλγόριθμο λύνοντας μία ακολουθία γραμμικών υποπροβλημάτων. Παρόμοια αποτελέσματα παρουσιάστηκαν από τους Chen et al. 1994.

Ένα κοινό χαρακτηριστικό σε όλους σχεδόν τους αλγόριθμους τύπου simplex είναι ότι μπορούμε να τους δούμε σαν μία διαδικασία που ακολουθεί κάποια μονοπάτια τα οποία οδηγούν στη βέλτιστη λύση. Αυτός ο αλγόριθμος διαφέρει ριζικά από τον πρωτεύοντα αλγόριθμο simplex (PSA) επειδή οι βασικές του λύσεις δεν είναι εφικτές. Έχει τονιστεί από τους Paparrizos et al. 2001, ότι η γεωμετρία του EPSA αποκαλύπτει ότι ο αλγόριθμος αυτός είναι πιο γρήγορος από

τη γνώριμη μέθοδο simplex, κάτι το οποίο επιβεβαιώθηκε από τα πρώτα συγκριτικά αποτελέσματα των πρώτων δυϊκών μεθόδων EPSA πάνω σε ειδικά δομημένα γραμμικά προβλήματα (Dosios et al 1997). Πρόσφατα, ο κ. Παπαρρίζος παρουσίασε υπολογιστικές μελέτες που δείχνουν ότι ο αλγόριθμος EPSA είναι πιο αποτελεσματικός από τον κλασικό πρωτεύοντα αλγόριθμο simplex PSA (Paparrizos et al. 2003).

3.6.4.1 Περιγραφή του αλγορίθμου

Θεωρούμε το πιο κάτω πρόβλημα γραμμικού προγραμματισμού

$$\begin{aligned} &\text{maximize} && c^T x \\ &\text{subject to} && Ax=b \\ & && x \geq 0, \end{aligned} \qquad (3.6.4.1.1)$$

όπου $A \in R^{m\times n}, c,x \in R^n, b \in R^m$ και T συμβολίζει transposition. Είναι βολικό να περιγράψουμε τον αλγόριθμο χρησιμοποιώντας την αναθεωρημένη μορφή του αλγόριθμου simplex. Υποθέτουμε ότι ο πίνακας A είναι πλήρους τάξης, rank(A) = m, $1 \leq m \leq n$. Θα χρησιμοποιήσουμε έναν παραδοσιακό τρόπο συμβολισμού γράφοντας B αντί για A_B και N αντί για A_N. Σε περιπτώσεις όπου τα B και N αντιστοιχούν σε υποσύνολο των δεικτών ή σε πίνακα, θα πρέπει αυτό να είναι ξεκάθαρο από τα συμφραζόμενα. Διαχωρίζοντας τον πίνακα ως
$A = (B \ N)$ και αντιστοίχως διαχωρίζοντας και διατάσσοντας τα x και c ως

$x=\begin{bmatrix} x_B \\ x_N \end{bmatrix}$, $c=\begin{bmatrix} c_B \\ c_N \end{bmatrix}$ (P.1) το πρόβλημα γράφεται

$$\begin{aligned} &\max && c_B^T x_B + c_N^T x_N \\ &\text{subject to} && Bx_B + Nx_N = b \end{aligned}$$

$$x_B, x_N \geq 0$$

Εδώ ο B είναι ένας $m \times m$ αντιστρέψιμος υποπίνακας του A που καλείται βασικός πίνακας. Οι στήλες του A που ανήκουν στον B καλούνται βασικές και οι υπόλοιπες μη-βασικές. Δοθέντος του βασικού πίνακα *B* η αντίστοιχη λύση $x_B = B^{-1}b$, $x_N = 0$ καλείται βασική λύση. Μία λύση $x=(x_B, x_N)$ είναι εφικτή εάν $x \geq 0$,αλλιώς καλείται μη εφικτή. Είναι γνωστό ότι η λύση του δυϊκού προβλήματος που αντιστοιχεί στη βάση B, δίνεται από τη σχέση $s=c-A^T w$ όπου $w^T=(c_B)^T B^{-1}$ είναι οι πολλαπλασιαστές simplex και s είναι οι δυϊκές χαλαρές μεταβλητές. Η αντίστοιχη βάση B καλείται δυϊκή εφικτή εάν είναι $s \geq 0$. Είναι γνωστό ότι $s_B=0$.

Σε κάθε επανάληψη, ο αλγόριθμος PSA εναλλάσσει μία στήλη του τρέχοντα B με μία στήλη του τρέχοντα μη βασικού N, δημιουργώντας με τον τρόπο αυτό μία νέα βάση. Γεωμετρικά, αυτό σημαίνει ότι ο PSA κινείται στα άκρα του πολύεδρου $P = \{x|Ax \leq b, x \geq 0\}$. Ένα τέτοιο μονοπάτι είναι γνωστό σαν μονοπάτι simplex. Από την άλλη μεριά ο EPSA δημιουργεί δύο μονοπάτια προς τη βέλτιστη λύση. Το ένα μονοπάτι είναι μη εφικτό (εξωτερικό) και το άλλο είναι εφικτό. Ο EPSA επομένως δεν χρειάζεται να προχωρήσει κινούμενος από άκρη σε άκρη στο πολύεδρο. Επομένως, μη έχοντας την υποχρέωση να κινούμαστε μέσα στην εφικτή περιοχή, μπορούμε να ακολουθήσουμε πιο σύντομα μονοπάτια προς τη βέλτιστη λύση. Παρατηρήστε ότι το κύριο μέρος της εργασίας σε κάθε επανάληψη σε αλγορίθμους τύπου simplex, αφιερώνεται στον προσδιορισμό του αντίστροφου πίνακα B^{-1}. Αυτός ο αντίστροφος όμως δεν είναι απαραίτητο να υπολογιστεί από την αρχή σε κάθε επανάληψη. Ο επόμενος

αντίστροφος $(B^{-1})^{(k+1)}$ μπορεί να υπολογιστεί από τον τρέχοντα αντίστροφο $(B^{-1})^{(k)}$ μέσω μιας απλής περιστροφής. Συγκεκριμένα έχουμε

$$(B^{-1})^{(k+1)}=E^{-1}(B^{-1})^{(k)}$$

Όπου ο E^{-1} είναι ο αντίστροφος του πίνακα ήτα (eta-matrix). Χρησιμοποιούμε την ακόλουθη σχέση για τον υπολογισμό του E^{-1}:

$$E^{-1}=I-\frac{1}{a_{rl}}(a_l-e_l)e_l^T$$

Στην πιο πάνω σχέση με a_{rl} συμβολίζουμε το οδηγό στοιχείο για την περιστροφή, η στήλη r καλείται οδηγός στήλη και η γραμμή l καλείται οδηγός γραμμή. Μία περιγραφή του πρωτεύοντα αλγόριθμου EPSA στην αναθεωρημένη του μορφή δίνεται πιο κάτω. Η πιο πάνω τεχνική καλείται μορφή γινομένου του αντιστρόφου.

Αλγόριθμος EPSA

Βήμα 0 (Ξεκίνημα).

Ξεκινάμε με μία εφικτή βασική διαμέριση (B,N).

Υπολογίζουμε τα διανύσματα και πίνακες B^{-1}, x_B, w, s_N. Βρίσκουμε τα σύνολα $P=\{j\in N : s_j<0\}$ και $Q=\{j\in N\ s_j\geq 0\}$ και επιλέγουμε ένα αυθαίρετο διάνυσμα $\lambda=(\lambda_1,\lambda_2,...,\lambda_{|P|})>0$, υπολογίζουμε το s_0 από τη σχέση $s_0=\sum_{j\in P}\lambda_j s_j$ και το διάνυσμα $d_B=-\sum_{j\in P}\lambda_j h_j$ με $h_j=B^{-1}A_{\cdot j}$

Βήμα 1 (Έλεγχος τερματισμού)

(i) (Έλεγχος βελτιστότητας) Αν $P=\varnothing$, ΤΕΛΟΣ. Το πρόβλημα (P.1) είναι βέλτιστο.

(ii) (Επιλογή της εξερχόμενης μεταβλητής). Αν $d_B \geq 0$, ΤΕΛΟΣ. Αν $s_0 = 0$, το πρόβλημα (P.1) είναι βέλτιστο. Αλλιώς, επιλέγουμε την εξερχόμενη μεταβλητή $x_{B[r]} = x_k$ από τη σχέση

$$\alpha = \frac{x_{B[r]}}{-d_{B[r]}} = \min\left\{\frac{x_{B[i]}}{-d_{B[i]}} : d_{B[i]} < 0\right\}$$

Αν $\alpha = +\infty$ το πρόβλημα (P.1) δεν είναι φραγμένο. Θέτουμε $y_B = x_B + \alpha d_B$

Βήμα 2 (Επιλογή της εισερχόμενης μεταβλητής).

Υπολογίζουμε τα διανύσματα $H_{rP} = (B^{-1})_{r.} A_{.P}$ και $H_{rQ} = (B^{-1})_{r.} A_{.Q}$. Επίσης βρίσκουμε τους λόγους θ_1 και θ_2 από τη σχέση

$$\theta_1 = \frac{-s_P}{h_{rP}} = \min\left\{\frac{-s_j}{h_{rj}} : h_{rj} > 0 \text{ and } j \in P\right\}$$

$$\theta_2 = \frac{-s_Q}{h_{rQ}} = \min\left\{\frac{-s_j}{h_{rj}} : h_{rj} < 0 \text{ and } j \in Q\right\}$$

Και προσδιορίζουμε τους δείκτες t_1 και t_2 έτσι ώστε $P(t_1) = p$ και $Q(t_2) = q$.Αν $\theta_1 \leq \theta_2$, θέτουμε $l = p$. Αλλιώς, θέτουμε $l = q$. Η μη βασική μεταβλητή x_l εισάγεται στη βάση.

Βήμα 3 (Περιστροφή).

Θέτουμε $B[r] = l$. Αν $\theta_1 \leq \theta_2$ θέτουμε $P = P \setminus \{l\}$ και $Q = Q \cup \{k\}$. Αλλιώς, θέτουμε $Q[t] = k$.

Χρησιμοποιώντας τη νέα διαμέριση (B, N), όπου $N = (P, Q)$, ενημερώνουμε τα διανύσματα και τους πίνακες B^{-1}, x_B, w, s_N , $d_B = y_B - x_B$.

Πηγαίνουμε στο βήμα 1.

3.6.5 Υπολογιστική μελέτη

Οι μέθοδοι ανανέωσης που περιγράφτηκαν στην παράγραφο 3 έχουν υλοποιηθεί πειραματικά. Σε αυτή την παράγραφο περιγράφουμε τα αριθμητικά πειράματα και παρουσιάζουμε τα υπολογιστικά αποτελέσματα, τα οποία δείχνουν την αποτελεσματικότητα των μεθόδων ανανέωσης σε ένα σύνολο προβλημάτων του NETLIB (www.netlib.org/lp). Η μεταγλώττιση έγινε στο MATLAB v7.01 SP2 με τις βασικές του ρυθμίσεις, και όλες οι εκτελέσεις έγιναν σε περιβάλλον MS Windows XP Pro σε υπολογιστή P4 3.06 GHz με 512 Mb RAM.

Ο πίνακας 1 παρουσιάζει τα στατιστικά των 16 μετρήσεων σε αλφαβητική σειρά. Η πρώτη στήλη περιλαμβάνει το όνομα του προβλήματος (1), ενώ οι άλλες δύο στήλες (2)-(3) περιέχουν το μέγεθος των προβλημάτων Στις γραμμές δεν περιλαμβάνεται η αντικειμενική συνάρτηση στη στήλη (4) εμφανίζεται το πλήθος των μη μηδενικών στοιχείων των A και c. Τέλος, η ποσοστιαία πυκνότητα περιέχεται στη στήλη (5).

Name (1)	m (2)	n (3)	nnz(A)+nnz(c) (4)	Density (5)
adlittle	56	97	465	8,56%
bandm	305	472	2659	1,85%
degen2	444	534	4449	1,88%
e226	223	282	2767	4,40%
israel	174	142	2358	9,54%
lotfi	153	308	1086	2,30%
sc105	105	103	281	2,60%
sc205	205	203	552	1,33%

scagr7	129	140	553	3,06%
sctap1	300	480	2052	1,42%
sctap3	1480	2480	10734	0,29%
share1b	118	225	1182	4,45%
share2b	96	79	730	9,62%
ship04l	402	2118	8450	0,99%
ship08s	778	2387	9501	0,51%
stocfor1	117	111	474	3,65%

Σύνολο 3.6.5.1: Στατιστικά των δοκιμασιών

Ο πίνακας 3.6.5.2 περιέχει πληροφορίες για την απόδοση των προαναφερόμενων μεθόδων. Οι στήλες του πίνακα 3.6.5.2 περιέχουν το όνομα του κάθε προβλήματος, το μέγεθος του προβλήματος, το πλήθος των επαναλήψεων, τον αριθμό των επαναλήψεων (niter) και το χρόνο CPU, και επιτάχυνση της cpu σε δευτερόλεπτα για τις μεθόδους PFI και MPFI.

			PFI		**MPFI**		
name	**m**	**n**	**niter**	**cpu**	**niter**	**cpu**	**speed-up**
adlittle	56	97	140	0,33	140	0,33	1,00
bandm	305	472	485	5,86	485	3,00	1,95
degen2	444	534	1869	54,42	1436	17,87	3,05
e226	223	282	693	3,04	693	1,92	1,58
israel	174	142	306	0,89	306	0,65	1,37
lotfi	153	308	565	1,73	565	1,01	1,71
sc105	105	103	111	0,14	111	0,08	1,75

sc205	205	203	260	0,97	260	0,71	1,37
scagr7	129	140	141	0,45	141	0,25	1,80
sctap1	300	450	676	2,56	704	6,31	0,41
sctap3	1480	2480	2421	42,02	2421	37,56	1,12
share1b	118	225	458	1,40	458	0,56	2,50
share2b	96	79	356	1,08	356	0,48	2,25
ship04l	402	2118	1282	6,05	1282	13,78	0,44
ship08s	778	2387	764	3,46	764	7,71	0,45
stocfor1	117	111	138	0,31	138	0,11	2,82
Μέση τιμή							**1,60**

Πίνακας 3.6.5.2: Αποτελέσματα υπολογισμών

Στα πιο πάνω αποτελέσματα μπορούμε να κάνουμε τις ακόλουθες παρατηρήσεις: (i) Η μέθοδος MPFI είναι στις περισσότερες περιπτώσεις πιο γρήγορη από την PFI (12 προς 16) και (ii) Χρησιμοποιώντας την MPFI ο συνολικός χρόνος εκτέλεσης ήταν 1.60 φορές πιο μικρός από ότι στην PFI.

3.6.5.1 Συμπεράσματα

Η προηγούμενη υπολογιστική μελέτη δείχνει ότι η τροποποίηση της μεθόδου PFI είναι 1.60 φορές πιο γρήγορη από ότι η ίδια η PFI για τα περισσότερα προβλήματα. Θεωρητικά, και οι δύο μέθοδοι απαιτούν το ίδιο πλήθος πολλαπλασιασμών και προσθέσεων.

ΚΕΦΑΛΑΙΟ 4

ΥΠΟΛΟΓΙΣΤΙΚΗ ΜΕΛΕΤΗ

4.1 Εισαγωγή

Η υπολογιστική μελέτη είναι μια μέθοδος σύγκρισης μεταξύ δυο αλγόριθμων για την επιλογή του βέλτιστου .Αυτό που πρέπει να ληφθεί υπ`όψιν κατά την μελέτη είναι η σύγκριση πάνω στα ίδια πρόβληματα .Αυτό το κεφάλαιο περιγράφει τα χαρακτηριστικά και την αρχιτεκτονική της υπολογιστικής μελέτης. Η παρούσα υπολογιστική μελέτη υπολογίζει το μέσο αριθμό επαναλήψεων και τον μέσο χρόνο CPU για τον αλγόριθμο simplex εξωτερικών σημείων (EPSA) και τον κλασσικό αλγόριθμο simplex (PSA) σε προβλήματα τυχαία και αραιά του γραμμικού προγραμματισμού. Συμπαιρένουμε ότι ο αλγόριθμος EPSA είναι περίπου δυο φορές πιο γρήγορος από το κλασσικό PSA αλγόριθμο σε γραμμικά προβλήματα μεσαίου μεγέθους . Αυτό μεταφράζεται σε αντίστοιχη οικονομία χρόνου CPU. Αν και ο υπολογιστικός φόρτος σε κάθε βήμα του EPSA είναι μεγαλύτερος από αυτόν για κάθε βήμα του PSA , η βελτίωση στον EPSA προέρχεται από το γεγονός ότι απαιτεί πολύ λιγότερες επαναλήψεις .Επίσης όσο το μέγεθος των προβλημάτων αυξάνεται και η πυκνότητα μειώνεται ο EPSA εμφανίζεται γρηγορότερος.

4.2 Χαρακτηριστικά τις Αρχιτεκτονικής της Υπολογιστικής Μελέτης

Ένας σωστός αλγόριθμος μοιάζει με ένα αιχμηρό μαχαίρι -- κάνει αυτό ακριβός που πρέπει χρησιμοποιώντας ελάχιστη προσπάθεια .Χρησιμοποιώντας ένα λάθος αλγόριθμο για την επίλυση ενός προβλήματος είναι σαν να θέλουμε να κόψουμε ένα κομμάτι κρέας με ένα κατσαβίδι .Το αποτέλεσμα μπορεί να είναι ικανοποιητικό ,όμως θα δαπανηθεί μεγαλύτερη προσπάθεια από αυτή που χρειάζεται και το αισθητικό αποτέλεσμα όχι και τόσο ικανοποιητικό.

Για να γίνει σύγκρισι μεταξύ δυο αλγορίθμων θα πρέπει να λάβουμε υπόψη δυο πράγματα : πρώτον τον μέσο όρο τον επαναλήψεων και δεύτερο το χρόνο CPU.

Ο αλγόριθμος για τον οποίο ο μέσος αριθμός επαναλήψεων και ο μέσος CPU χρόνος είναι μικρότερος είναι και ο πιο αποτελεσματικός αλγόριθμος.

Στην υπολογιστική μελέτη, γίνεται σύγκριση μεταξύ δυο αλγορίθμων, του κλασσικού αλγόριθμου simplex και του αλγόριθμου των εξωτερικών σημείων .

Και οι δυο αυτοί αλγόριθμοι PSA και EPSA ξεκινούν με ένα βασικό εφικτό σημείο , όμως το EPSA υπολογίζει λύσεις οι οποίες δεν είναι εφικτές και γι`αυτό ο αλγόριθμος ονομάζεται αλγόριθμος εξωτερικών σημείων.

Στην υπολογιστική μας μελέτη χρησιμοποιούμε δυο διαφορετικές πυκνότητες για την λύση των προβλημάτων: 5% και 10%.Για κάθε μια από αυτές τις σχέσεις υπάρχουν άλλες τρεις διαφορετικές κλασείς διαστάσεων : n×n, n×2n και 2n×n.

Η πρώτη κλάση n×n περιλαμβάνει οκτώ διαφορετικές τιμές του n που αντιστοιχούν στις τιμές n = 300, 350, 400, 450, 500, 600, 700, 800. Καθε μια από αυτές τις τιμές περιέχουν 10 τυχαία αραιά προβλήματα γραμμικού προγραμματισμού . Οι δυο άλλες κλάσεις nx2n και 2nxn, περιλαμβάνουν τρεις διαφορετικές τιμές που αντιστοιχούν στις τιμές n = 300, 350, 400.

Ο Πίνακας 4.2.1: Περιλαμβάνει στατιστικά στοιχεία των προβλημάτων που χρησιμοποιήθηκαν σε αυτήν την υπολογιστική μελέτη.

Πεδίο τιμών του A	[-700 1800]
Πεδίο τιμών του A	[1 500]
Πεδίο τιμών του A	[10 100]
Συνολικός αριθμός των LP	280

Πίνακας 4.2.1 : Στατιστικά στοιχεία για τυχαία, αραιά προβλήματα.

Τα αριθμητικά μας πειράματα πραγματοποήθηκαν σε προσωπικό υπολογιστή (PC) με 2000 MHZ, Pentium 4 επεξεργαστή RAM 512 MB και Windows XP. Οι αλγόριθμοι προγραμματίστηκαν στη γλώσσα προγραμματισμού C.
Τα τυχαία γραμμικά προβλήματα υπολογίστηκαν στο MATLAB [Samaras 2000]
Τα γραμμικά προβλήματα που χρησιμοποιήθηκαν σ`αυτήν την υπολογιστική μελέτη έχουν την παρακάτω μορφή:

$$\max \quad c^T x$$
$$s.t.: \quad Ax \le b$$
$$x \ge 0$$

όπου $A \in R^{m+n}$, $c, x \in R^n$ and $b \in R^m$

4.3. Παρουσίαση αποτελεσμάτων

Στους πίνακες που ακολουθούν, για κάθε μέγεθος και πυκνότητα των προβλημάτων αναφέρονται κάποιες στατιστικές που υπολογίστηκαν στην

υπολογιστική μελέτη καθώς και πληροφορίες για την απόδοση των δυο αλγόριθμων, του PSA και του EPSA.

Οι στήλες των πίνακων 4.3.1 - 4.3.6 περιέχουν τις διαστάσεις των προβλημάτων, τον αριθμό των μη μηδενικών στοιχείων του πίνακα συντελεστών (εκτός από τα διανύσματα κόστους και δεξιού μέρους), τον μέσο αριθμό επαναλήψεων και το μέσο χρόνο CPU.

n x n	nnz	PSA		EPSA	
		niter	CPU	niter	CPU
300 × 300	4390.0	3683.9	31.7053	2090.0	18.7477
350 × 350	6139.5	4908.3	58.6021	2945.5	34.4852
400 × 400	7807.2	6276.2	97.9857	2889.7	56.9017
450 × 450	10097.8	8351.5	167.0880	3574.3	91.6215
500 × 500	12195.8	10086.5	250.6490	4212.6	134.2010
600 × 600	17546.0	14713.3	540.0090	5445.5	261.4220
700 × 700	23896.4	20107.6	1038.44 0	6930.8	463.7710
800 × 800	31213.1	31310.7	2212.4300	8708.9	775.7790

Πίνακα 4.3.1: Υπολογιστικά αποτελέσματα για αραιά γραμμικά προγράμματα $n \times n$ με πυκνότητα *5%*

n x n	nnz	PSA		EPSA	
		niter	CPU	niter	CPU
300 × 300	8573.6	3370.1	31.6330	2022.9	20.0335
350 × 350	11664.1	4448.5	56.6681	2336.1	34.3671
400 × 400	15216.2	5173.4	89.0898	2473.8	53.4746

450 × 450	21208.5	7478.7	163.3977	3579.7	97.8625
500 × 500	23780.1	8424.7	272.2390	3778.9	129.3933
600 × 600	34251.0	12929.5	543.6208	5405.3	274.6750
700 × 700	46633.3	17158.6	1001.8940	6752.5	484. 2333
800 × 800	60911.2	22861.1	1965.3400	8152.7	778.2910

Πίνακα 4.3.2: Υπολογιστικά αποτελέσματα για αραιά γραμμικά προγράμματα $n \times n$ με πυκνότητα *10%*

n x 2n	nnz	PSA		EPSA	
		niter	CPU	niter	CPU
300 × 600	8787.3	5980.2	98.6748	4334.5	82.5895
350 × 700	11945.1	8599.7	197.7200	5472.4	155.1030
400 × 800	15635.8	10970.4	335.3570	6571.1	256.8260

Πίνακα 4.3.3: Υπολογιστικά αποτελέσματα για αραιά γραμμικά προγράμματα $n \times 2n$ με πυκνότητα *5%*

n x 2n	nnz	PSA		EPSA	
		niter	CPU	niter	CPU
300 × 600	17117.5	5249.4	89.4705	3911.4	77.5788
350 × 700	23295.2	7000.4	170.1536	4830.2	141.2967
400 × 800	30440.6	8979.3	294.852	6290.4	231.7487

Πίνακας 4.3.4: Υπολογιστικά αποτελέσματα για αραιά γραμμικά προγράμματα $n \times 2n$ με πυκνότητα *10%*

2n x n	nnz	PSA		EPSA	
		niter	CPU	niter	CPU
600 ×300	8781.2	4228.3	105.287	232.75	66.0007
700× 350	12286.3	6117.2	206.368	3015.2	122.935
800 × 400	15611.6	8580.4	401.718	3675.1	193.491

Πίνακας 4.3.5: Υπολογιστικά αποτελέσματα για αραιά γραμμικά προγράμματα $2n \times n$ με πυκνότητα *5%*

2n x n	nnz	PSA		EPSA	
		niter	CPU	niter	CPU
600 ×300	17135.2	5022.5	133.142	2464.5	75.7136
700× 350	23325.8	7058.2	278.003	3031.9	132.701
800 × 400	30468.7	9302.3	443.655	3680.5	209.991

Πίνακας 4.3.6: Υπολογιστικά αποτελέσματα για αραιά γραμμικά προγράμματα $2n \times n$ με πυκνότητα *10%*

Για να γίνει περισσότερο σαφής η ανωτερότητα του αλγόριθμου EPSA σε σχέση με τον αλγόριθμο PSA παρέχουμε στους πίνακες 5.3.7 – 5.3.12 τους λόγους των μέσων αριθμών επαναλήψεων και των μέσων χρόνων CPU για κάθε κλάση πυκνότητας και τιμή του n.

n x n	niter PSA/EPSA	CPU PSA/EPSA
300 × 300	1.7626	1.6912
350 × 350	1.6664	1.6993

400 × 400	2.1719	1.7220
450 × 450	2.3365	1.8237
500 × 500	2.3944	1.8677
600 × 600	2.7019	2.0657
700 × 700	2.9012	2.2391
800 × 800	3.5953	2.8519
Mean value	**2.4413**	**1.9951**

Πίνακας 4.3.7: *Λόγοι μ.ο. επαναλήψεων και μέσων CPU χρόνων για αραιά γραμμικά προγράμματα n×n με πυκνότητα* 5%

Από το πίνακα 4.3.7 βλέπουμε τις διαστάσεις 800x800 πυκνότητας 5%, ο EPSA είναι 3.5953 φορές γρηγορότερος από τον PSA όσον αφορά τις επαναλήψεις και λύνει τα προβλήματα 2.8519 φορές γρηγορότερα από τον PSA όσον αφορά το CPU χρόνο.

n x n	niter PSA/EPSA	CPU PSA/EPSA
300 × 300	1.6660	1.5790
350 × 350	1.9042	1.6489
400 × 400	2.0913	1.6660
450 × 450	2.0892	1.6697
500 × 500	2.2294	1.7562
600 × 600	2.3920	1.9791
700 × 700	2.5411	2.0690
800 × 800	2.8041	2.5252
Mean value	**2.2147**	**1.8616**

Πίνακα 4.3.8: *Λόγοι μ.ο. επαναλήψεων και μέσων χρόνων για αραιά γραμμικά προγράμματα n×n με πυκνότητα* 10%

$2n$ x n	niter PSA/EPSA	CPU PSA/EPSA
600×300	3.7728	1.5962
700×350	4.0748	1.6787
800×400	4.2479	2.0762
Mean value	**4.0318**	**1.7837**

Πίνακας 4.3.9: *Λόγοι μ.ο. επαναλήψεων και μέσων χρόνων για αραιά γραμμικά προγράμματα* $2n \times n$ *με πυκνότητα* 5%

Στον πίνακα 4.3.9, βλέπουμε πως τις διαστάσεις 800 × 400 και πυκνότητα 5% ο EPSA είναι 4.2479 φορές γρηγορότερος από τον PSA όσον αφορά τις επαναλήψεις και λύνει τα προβλήματα 2.0762 φορές γρηγορότερα από τον PSA όσον αφορά το CPU χρόνο.

$2n$ x n	niter PSA/EPSA	CPU PSA/EPSA
600×300	2.0379	1.7585
700×350	2.3280	2.0950
800×400	2.5275	2.1194
Mean value	**2.2978**	**1.9910**

Πίνακα 4.3.10 : *Λόγοι μ.ο. επαναλήψεων και μέσων χρόνων για αραιά γραμμικά προγράμματα* $2n \times n$ *με πυκνότητα* 10%

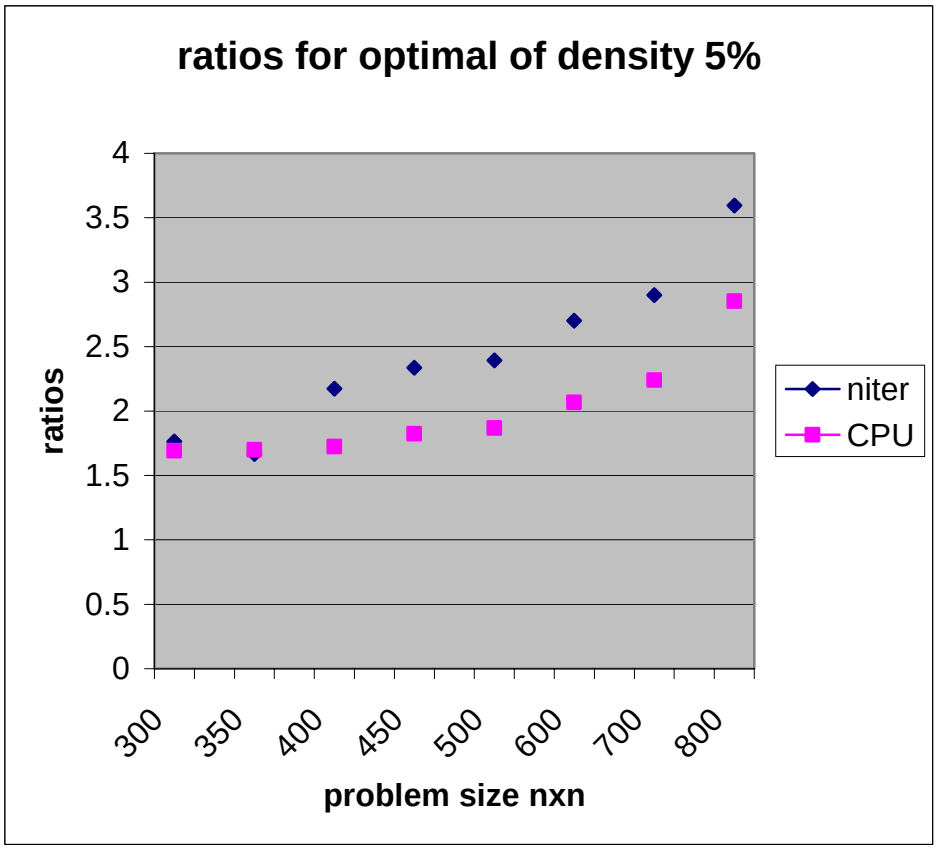
n x 2*n*	niter PSA/EPSA	CPU PSA/EPSA
300 × 600	2.0273	1.1948
350 × 700	2.1828	1.2748
400 × 800	2.3795	1.3058
Mean value	**2.1965**	**1.2585**

Πίνακα 4.3.11: *Λόγοι μ.ο. . επαναλήψεων και μέσων χρόνων για αραιά γραμμικά προγράμματα n×2n με πυκνότητα* 5%

n x 2*n*	Niter PSA/EPSA	CPU PSA/EPSA
300 × 600	1.3421	1.1533
350 × 700	1.4493	1.2042
400 × 800	1.4275	1.2729
Mean value	**1.4063**	**1.2101**

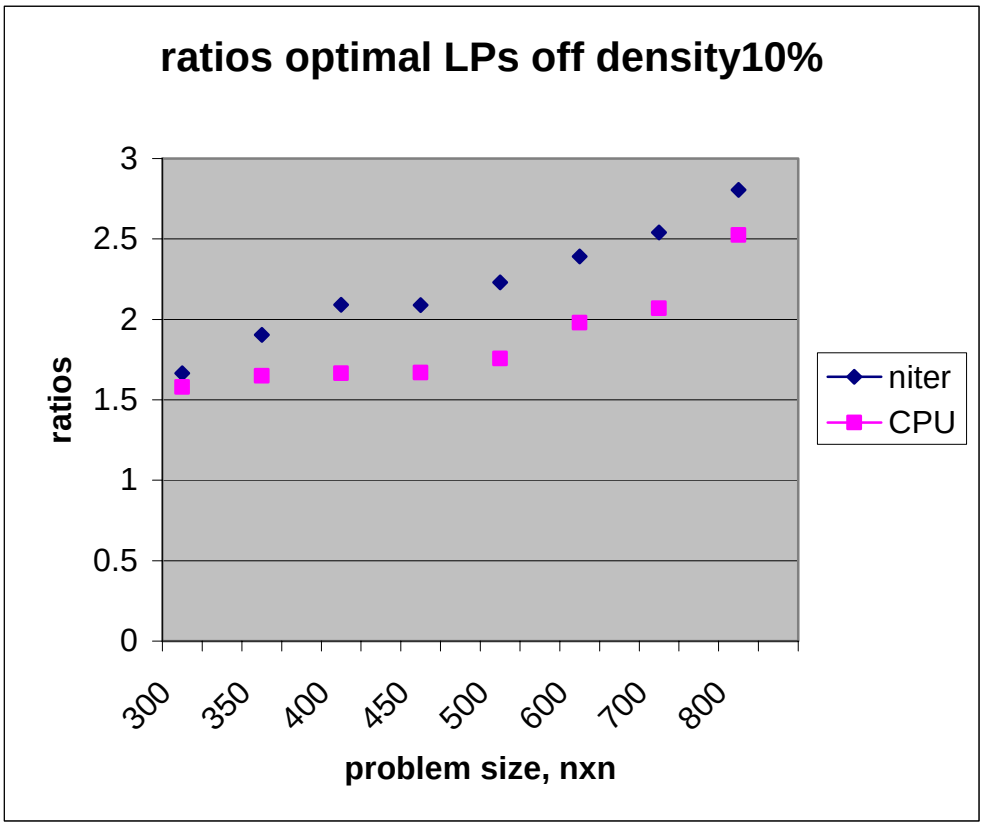
Πίνακα 4.3.12: *Λόγοι μ.ο. . επαναλήψεων και μέσων χρόνων για αραιά γραμμικά προγράμματα n×2n με πυκνότητα* 10%

Κάνουμε τώρα γραφική παράσταση των λόγων που φθίνονται στον πίνακα 4.3.7. Η γραφική παράσταση φθίνεται στην εικόνα 4.3.1 και αφορά την κλάση $n \times n$ με πυκνότητα 5%.



***Εικόνα* 4.3.1 :** *Λόγοι μ.ο. . επαναλήψεων και μέσων χρόνων της PSA σε σχεση με τον EPSA για βέλτιστα αραιά προβλήματα* $n \times n$ *με πυκνότητα* 5%

Κάνουμε τώρα γραφική παράσταση των λόγων που φθίνονται στον πίνακα 4.3.6. Η γραφική παράσταση φθίνεται στην εικόνα 4.3.2 και αφορά την κλάση n × n με πυκνότητα 10%.



***Εικόνα* . 4.3.2 :** *Λόγοι μ.ο. επαναλήψεων και μέσων χρόνων του PSA σε σχεση με τον EPSA για βέλτιστα αραιά προβλήματα* $n \times n$ *με πυκνότητα 10 %*

Από τα παραπάνω γραφήματα βλέπουμε ότι όσο αυξάνουν οι διαστάσεις του προβλήματος η ανωτερότητα του EPSA σε σχέση με τον PSA αυξάνεται. Μπορούμε να δούμε για παράδειγμα στην περίπτωση αραιών Γραμμικών Προβλημάτων με διαστάσεις n × n ότι όσο αυξάνεται η πυκνότητα του προβλήματος, η ανωτερότητα του αλγόριθμου EPSA σε σχέση με τον PSA αυξάνεται. Αυτό το συμπέρασμα είναι σημαντικό διότι όλα τα πραγματικά προβλήματα είναι εξαιρετικά αραιά . Ιδιαίτερα σε διαστάσεις 800x800 πυκνότητας

5%, ο EPSA είναι 3.5953 φορές πιο γρήγορος από τον PSA όσον αφορά τις επαναλήψεις και λύνει τα προβλήματα 2.8519 φορές γρηγορότερα από τον PSA όσον αφορά τον χρόνο CPU. Αν και ο υπολογιστικός φόρτος που απαιτείται σε κάθε βήμα του EPSA καταναλώνει περισσότερο χρόνο σε σύγκριση με αυτόν κάθε βήματος του PSA, η βελτίωση του EPSA προέρχεται από το γεγονός ότι απαιτεί πολύ λιγότερες επαναλήψεις από τον PSA. Επιπλέον όσο οι διαστάσεις του προβλήματος αυξάνονται και η πυκνότητα μειώνεται ο EPSA γίνεται γρηγορότερος.

ΚΕΦΑΛΑΙΟ 5

ΠΑΡΑΛΛΗΛΗ ΕΠΕΞΕΡΓΑΣΙΑ

5.1 Εισαγωγή

Η απαίτηση για γρηγορότερους και πιο αποτελεσματικούς υπολογισμούς σε επιστημονικές και εμπορικές εφαρμογές έχει αυξηθεί σημαντικά τα τελευταία χρόνια.

Φυσικοί περιορισμοί και υψηλά κόστη καθιστούν αδύνατη την αύξηση της ταχύτητας των επεξεργαστών πέρα από συγκεκριμένα όρια . Για να ξεπεραστούν αυτές οι δυσκολίες νέες αρχιτεκτονικές έχουν εμφανιστεί εισάγοντας την παράλληλη επεξεργασία . Στις μέρες μας διαφορετικοί τύποι μηχανών με υψηλές επιδόσεις είναι διαθέσιμες και στηρίζονται στην συγκέντρωση πολλών επεξεργαστών μαζί.Ανάμεσα σ`αυτά τα συστήματα τα σημαντικότερα είναι (Lau 1996) : διανυσματικοί πολυεπεξεργαστές (ένας μικρός αριθμός από διανυσματικοί επεξεργαστές υψηλών επιδόσεων); μαζικά παραλληλοί επεξεργαστές (εκατοντάδες έως εκατομμύρια επεξεργαστών συνδεδεμένων μεταξύ τους με κοινή κατανεμημένη μνήμη); και υπολογιστές σε δίκτυο (σταθμοί εργασίας συνδεδεμένοι με ένα μεσαίας ή υψηλών ταχυτήτας δίκτυο, οι οποίοι εργάζονται σαν μια παράλληλη εικονική μηχανή υψηλών επιδόσεων).

Ο παραλληλισμός οδηγεί στην ανάγκη για νέους αλγόριθμους, ειδικά σχεδιασμένους ώστε να εκμεταλλεύονται σύγχρονες διαδικασίες, λόγω του ότι πολλές φορές οι βέλτιστες παράλληλες λύσεις δεν επιτυγχάνονται με απλές τροποποιήσεις των ακολουθιακών αλγορίθμων (Foster 1995; Kumar et al 1994). Σε αντίθεση με τα ακολουθιακά προγράμματα, οι παράλληλοι αλγόριθμοι εξαρτώνται ισχυρά από την αρχιτεκτονική υπολογιστών για τους οποίους έχουν

σχεδιαστεί . Προγράμματα τα οποία υλοποιούν αυτούς τους αλγορίθμους έχουν αναπτυχθεί και εκτελεστεί σε παράλληλα υπολογιστικά περιβάλλοντα τα οποία αποτελούνται από έναν παράλληλο υπολογιστή και το σχετικό λογισμικό (Chapman et al 1999; Wolf and Kramer-Fuhrmann 1996). Το λογισμικό αποτελείται από παράλληλα προγραμματιστικά εργαλεία, εργαλεία απόδοσης και σχετικούς μεταγλωττιστές καθώς και ορισμένες βιβλιοθήκες που αναπτυχθήκαν για την επίλυση ειδικών κατηγοριών προβλημάτων (Blackford et al 1997; Merlin et al 1999; Nieplocha et al 1994; Plastino et al 1999). Τα προγραμματιστικά εργαλεία τυπικά παρέχουν υποστήριξη για την ανάπτυξη προγραμμάτων, τα οποία αποτελούνται από διεργασίες που ανταλλάσουν πληροφορίες κατά την διάρκεια των εκτελέσεων.

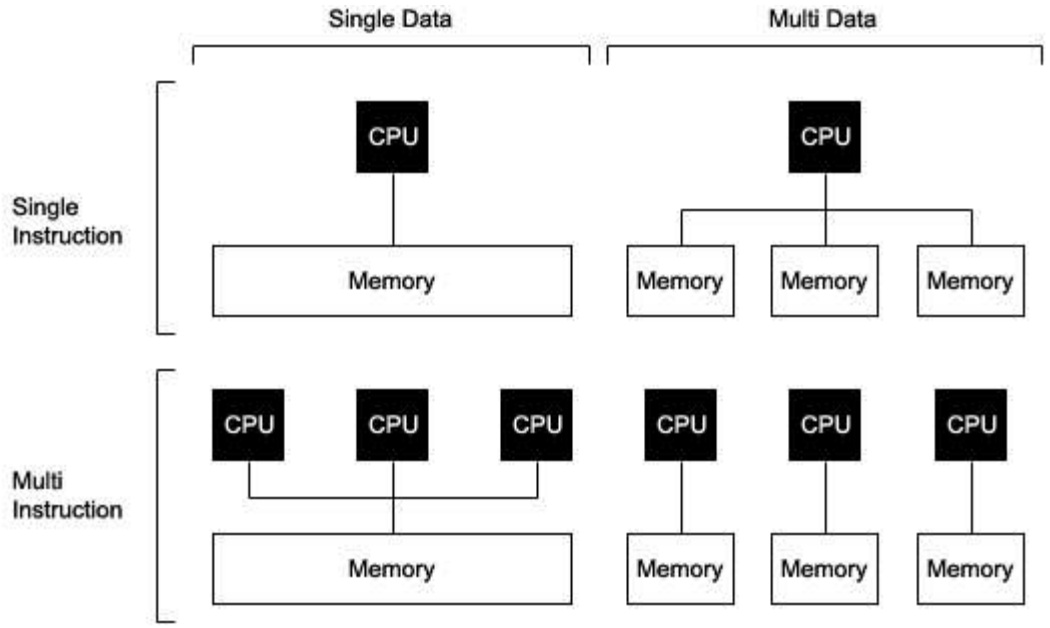
Η απόδοση των συστημάτων που αναπτύσσονται με προγραμματιστικά εργαλεία και η διαθέσιμη υποστήριξη για επικοινωνία και συγχρονισμό μεταξύ των τμημάτων του παράλληλου προγράμματος, είναι μερικά από τα σημαντικά σενάρεια του παράλληλου προγραμμάτισμου. Το προγραμματιστικό εργαλείο θα έπρεπε επίσης να είναι αντάξιο του προγραμματιστικού μοντέλου σύμφωνα με το οποίο υλοποιείται ο παράλληλος αλγόριθμος.

Η ευκολία και ο χρόνος για την ανάπτυξη ενός προγράμματος είναι πολλές φορές εξαιρετικά κρίσιμα .Τα εργαλεία μπορούν να βοηθήσουν τον προγραμματιστή ώστε να καταλάβει και να βελτιώσει παράλληλα προγράμματα με το να παρακολουθεί την εκτέλεση ενός παράλληλου προγράμματος και την παραγωγή δεδομένων τα οποία αναλύονται ώστε να εντοπιστούν και να γίνουν αντιληπτές περιοχές χαμηλής απόδοσης.

Η διόρθωση παραλλήλων προγραμμάτων είναι δύσκολο πρόβλημα. Παράλληλα εργαλεία διόρθωσης έχουν κατασκευαστεί και για παράλληλα προγράμματα και είναι συνήθως ιδιαίτερα αποτελεσματικά.

5.2 Κατηγορίες Παράλληλων Υπολογιστών.

Η πιο δημοφιλής γενική κατηγοριοποίηση παραλλήλων αρχιτεκτονικών έχει προταθεί από τον Flynn (1972). Οι τέσσερις κατηγορίες αρχιτεκτονικών περιγράφονται στην εικόνα 6.1.



Εικόνα 5.2.1. Αρχιτεκτονικές παράλληλων υπολογιστών

Ονομάζονται :

- SISD : (Single Instruction Single Data) μια CPU με μια μονάδα μνήμης
- SIMD: (Single Instruction Multi Data) ίδια εντολή σε πολλές μνήμες.
- MISD: (Multi Instruction Single Data) πολλές CPU με μια κοινή μονάδα μνήμης
- MIMD:(Multi Instruction Multi Data) μια CPU με δικές τους μνήμες η καθεμιά.

Οι προσωπικοί υπολογιστές είναι συνήθως SSID, εκτός από τους PC με πολλούς επεξεργαστές που είναι συνήθως MISD, οι SIMD αρχιτεκτονικές ονομάζονται διανυσματικοί υπολογιστές διότι μια εργασία μπορεί να γίνει ταυτόχρονα για διάφορα δεδομένα, όπως για παράδειγμα είναι οι πράξεις σε διανύσματα και πίνακες (κάτι που είναι το πλεονέκτημα των SIMD υπολογιστών). Πολλοί Cray υπολογιστές είναι βασισμένοι σε SIMD αρχιτεκτονικές. Το μεγαλύτερο μέρος των παραλλήλων μηχανών είναι MIMD όπου τα δεδομένα και οι εντολές είναι κατανεμημένα με τη μέθοδο της ανταλλαγής μυνημάτων, αποθηκεύονται σύμφωνα με την οποία τα δεδομένα επεξεργαζόνται τοπικά και τελικά αποστέλλονται πίσω. Μια άλλη προσέγγιση για παροχή δεδομένων ονομάζεται MISD όπου τα δεδομένα δε χρειάζεται να προωθούνται λόγω του ότι όλοι οι κόμβοι έχουν εύκολη πρόσβαση στην μνήμη. Παρ'όλα αυτά η αύξηση της πολυπλοκότητας οδηγεί σε αύξηση της προσπάθειας συντονισμού απλά επειδή πρόσβαση σε συγκεκριμένες περιοχές μνήμης την ίδια χρονική στιγμή από 2 διαφορετικές CPU κάνουν πιο πολύπλοκη τη διαδικασία εξασφάλισης της ακεραιότητας των δεδομένων μια διαδικασία που πρέπει να πραγματοποιείται μόνο από εξειδικευμένο λογισμικό ή υλικό.

5.3 Έννοιες παράλληλου προγραμματισμού

Τα παράλληλα προγράμματα αποτελούνται από έναν αριθμό ταυτόχρονων διαδικασιών. Αυτές οι διαδικασίες εκτελούνται σε παράλληλες μηχανές βασισμένες σε διαφορετικούς τρόπους οργάνωσης μνήμης: κοινή και κατανεμημένη (Elias 1995; Kumar et al 1994). Σε μηχανές με κοινή μνήμη (Protic et al 1998), όλοι οι επεξεργαστές μπορούν να προσπελάσουν όλες τις περιοχές μνήμης, όπως φαίνεται στην Εικόνα5.3.1. Η επικοινωνία μεταξύ των διεργασιών επιτυγχάνεται με ανάγνωση και εγγραφή των εργασιών που πραγματοποιούνται από τις διεργασίες στην κοινή μνήμη. Το κύριο πλεονέκτημα αυτής της τακτικής είναι ότι η πρόσβαση σε δεδομένα είναι πολύ γρήγορη και η επικοινωνία μεταξύ των

επεξεργαστών γίνεται μέσω της κοινής μνήμης. Παρ'όλα αυτά η επεκτασιμότητα είναι περιορισμένη από τον αριθμό των καναλιών επικοινωνίας μεταξύ μνήμης και επεξεργαστών, πράγμα που μπορεί να οδηγήσει σε χαμηλή απόδοση λόγω σύγχρονων αιτήσεων πρόσβασης. Για την αντιμετώπιση του προβλήματος οι επεξεργαστές ενισχύονται με τοπική μνήμη. Αυτή η μνήμη αποθηκεύει τον κώδικα που εκτελείται από τον επεξεργαστή και τις δομές δεδομένων που δεν είναι χρήσιμες σε κάποιον άλλο. Οι καθολικές δομές δεδομένων αποθηκεύονται στην κοινή μνήμη όπως φαίνεται στην Εικόνα 5.3.2. Μια επέκταση αυτού του είδους αρχιτεκτονικής είναι η αφαίρεση οποιασδήποτε κοινής φυσικής μνήμης όπως φαίνεται στην Εικόνα 5.3.3. Και στα δυο μοντέλα οι αναφορές ενός επεξεργαστή στη μνήμη ενός άλλου επεξεργαστή πραγματοποιούνται από το υλικό που συνδέει τη μνήμη και τους επεξεργαστές. Εφόσον κάθε πρόσβαση στην τοπική μνήμη είναι αρκετά γρήγορη, αυτό το μοντέλο βελτιώνει την ταχύτητα ανάκτησης δεδομένων. Ένα μειονέκτημα είναι ότι ο έλεγχος της πρόσβασης στη μνήμη, απαραίτητος για τη διατήρηση ακεραιότητας των δεδομένων, πρέπει να γίνει από τον προγραμματιστή.

Στις μηχανές με κατανεμημένη μνήμη, αυτή κατανέμεται και μεταξύ των επεξεργαστών. Κάθε επεξεργαστής έχει πρόσβαση μόνο στο δικό του κομμάτι μνήμης, Εικόνα 5.3.4. Η επικοινωνία μεταξύ των διεργασιών οι οποίες λαμβάνουν χώρα σε διαφορετικούς επεξεργαστές γίνεται με μηνύματα που ανταλλάσσονται μέσω των υπαρχόντων καναλιών επικοινωνίας. Τα τοπικά δίκτυα υπολογιστών αποτελούν μια καλή αναλογία των παραπάνω, και η σημασία τους αυξάνει συνεχώς. Παράλληλα η επεκτασιμότητα και η διαχείριση λαθών σε τέτοια δίκτυα, παρουσιάζουν πολύ καλή αναλογία κόστους-απόδοσης σε σχέση με άλλες αρχιτεκτονικές. Αυτό έχει οδηγήσει στη ολοένα αυξανόμενη χρήση *clusters,* τα οποία είναι δίκτυα μεγάλων ταχυτήτων που συνδέουν το υλικό ενός υπολογιστή.

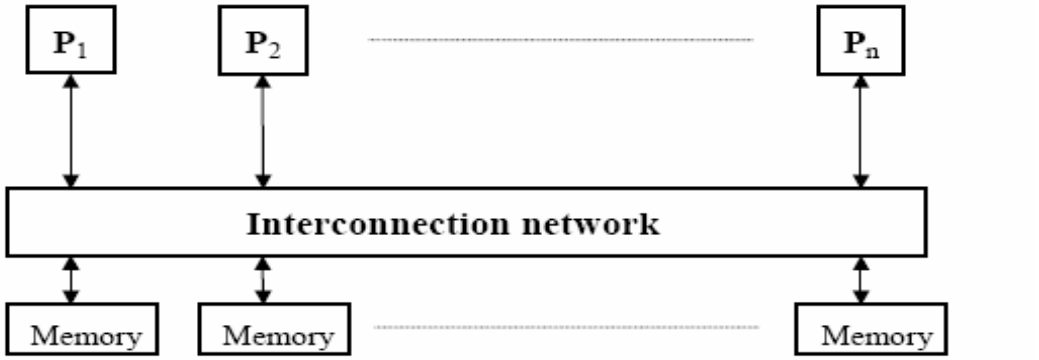


Figure 5.3.1: Shared memory architecture

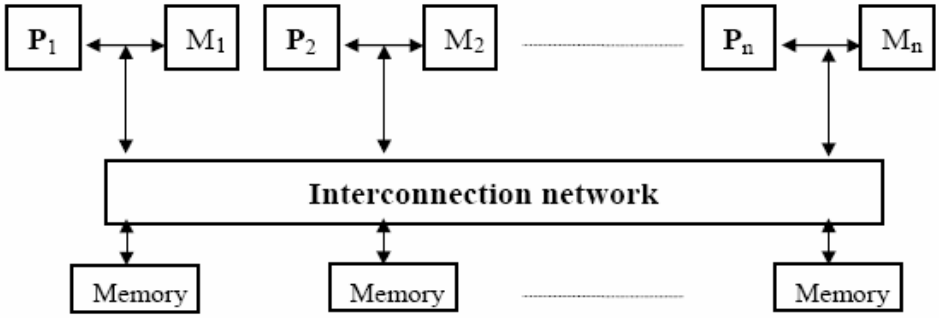


Figure 5.3.2: Shared memory architecture with local memory for processors

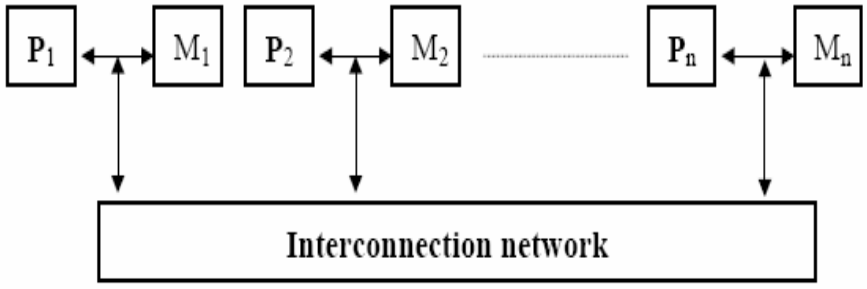


Figure 5.3.3: Shared memory architecture with **only** local memory for processors

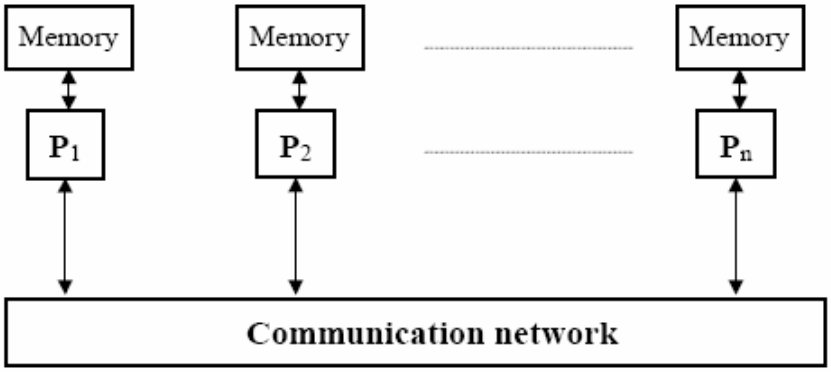


Figure 5.3.4: Distributed memory architecture

Δυο βασικοί τρόποι επικοινωνίας χρησιμοποιούνται για την ανταλλαγή πληροφοριών μεταξύ παράλληλων διεργασιών. Οι διεργασίες μπορεί να επικοινωνούν με πρόσβαση σε κοινή μνήμη (shared memory), ή με ανταλλαγή μηνυμάτων (message passing) (Andrews and Schneider 1983). Στο μοντέλο της κοινής μνήμης, οι διεργασίες μοιράζονται μια ομάδα διευθύνσεων μνήμης, τις οποίες προσπελαύνουν ασύγχρονα. Μηχανισμοί όπως locks και semaphores (Andrews and Schneider 1983; Dijkstra 1968; and Vahalia 1996) χρησιμοποιούνται για το συντονισμό της διαδικασίας. Αυτό το μοντέλο επικοινωνίας μπορεί να χρησιμοποιηθεί απ' ευθείας σε διαφορετικές παράλληλες μηχανές κοινής μνήμης αλλά και να προσομοιωθεί σε παράλληλες μηχανές με κατανεμημένη μνήμη. Στην τελευταία περίπτωση ο χρήστης βλέπει την κοινή μνήμη σαν μια οντότητα αν και αυτή είναι κατανεμημένη σε διαφορετικές μηχανές. Ένας τρόπος για την υλοποίηση του παραπάνω μηχανισμού είναι να χρησιμοποιηθούν διεργασίες, οι οποίες (i) μεταφράζουν κάθε διεύθυνση της κοινής μνήμης στη φυσική της τοποθεσία, και (ii) είναι υπεύθυνες για την αποστολή μηνυμάτων σε άλλες διεργασίες για ανάγνωση και εγγραφή δεδομένων.

Στο μοντέλο ανταλλαγής μηνυμάτων, οι επεξεργαστές επικοινωνούν με ανταλλαγή μηνυμάτων με δυο τρόπους (blocking και non-blocking mode). Στην πρώτη περίπτωση, η διεργασία που στέλνει το μήνυμα σταματά τις ενέργειές της έως την παραλαβή του μηνύματος. Στη δεύτερη περίπτωση η εκτέλεση συνεχίζεται χωρίς διακοπές. Στο blocking μοντέλο, η διαδικασία σταματά τις ενέργειές της έως την παραλαβή σχετικού μηνύματος, ενώ στο non-blocking μοντέλο γίνεται απλός έλεγχος για την παραλαβή μηνύματος και η εκτέλεσή της συνεχίζεται σε διαφορετική περίπτωση. Το μοντέλο ανταλλαγής μηνυμάτων μπορεί να χρησιμοποιηθεί άμεσα στους υπολογιστές κατανεμημένης μνήμης. Επίσεις μπορεί να προσομοιωθεί πολύ εύκολα. (Tanenbaum 1992).

Πολλά παράλληλα προγραμματιστικά εργαλεία είναι ουσιαστικά ακολουθιακές γλώσσες προγραμματισμού με την ενίσχυση ειδικών κλήσεων του συστήματος. Αυτές οι κλήσεις παρέχουν την ικανότητα ανταλλαγής μηνυμάτων, συγχρονισμό διαδικασιών, δημιουργία διεργασιών, αμοιβαίο αποκλεισμό και λοιπές λειτουργίες. Ορισμένα από τα εργαλεία αυτά παρουσιάζονται στη συνέχεια.

5.4 Παράλληλα Προγραμματιστικά Εργαλεία

Πολλά προγραμματιστικά εργαλεία είναι διαθέσιμα για την υλοποίηση παραλλήλων προγραμμάτων, κάθε ένα από τα οποία είναι χρήσιμο σε συγκεκριμένες εφαρμογές. Κάθε επιλογή παράλληλου εργαλείου εξαρτάται από τα χαρακτηριστικά του προβλήματος. Κάποια εργαλεία είναι καλλίτερα για αλγορίθμους αριθμητικής ανάλυσης ενώ κάποια άλλα είναι χρήσιμα σε περιπτώσεις δυναμικής δημιουργίας διεργασιών και μη κανονικών δομών δεδομένων. Έχουν επιλεγεί ορισμένα εργαλεία τα οποία βρίσκονται σε ικανοποιητικό στάδιο αξιοπιστίας και έχουν χρησιμοποιηθεί σε εφαρμογές, οι οποίες καλύπτουν ένα ευρύ φάσμα.

5.4.1 PVM

Οι Παράλληλες Ιδεατές Μηχανές (*Parallel Virtual Machine*) (Geist et al 1994) είναι μια βιβλιοθήκη ανταλλαγής μηνυμάτων με ευρεία χρήση, η οποία δημιουργήθηκε για την υποστήριξη της ανάπτυξης κατανεμημένων και παράλληλων προγραμμάτων σε δίκτυα ετερογενών υπολογιστών.

Ένα PVM πρόγραμμα αποτελείται από ένα σύνολο διεργασιών που επικοινωνούν μέσω μιας ιδεατής μηχανής ανταλλαγής μηνυμάτων. Ένα αρχείο ρυθμίσεων που δημιουργείται από το χρήστη καθορίζει ποιες πραγματικές μηχανές θα αποτελούν την ιδεατή. Μια διεργασία συντονισμού που εκτελείται σε κάθε μια από τις παραπάνω μηχανές ελέγχει την ανταλλαγή μηνυμάτων μεταξύ τους. Αυτές είναι οι υπο-ρουτίνες αρχικοποίησης και τερματισμού διεργασιών, ανταλλαγής μηνυμάτων, δημιουργία ομάδων, συντονισμού επικοινωνίας μεταξύ διεργασιών, συγχρονισμού και δυναμικής αλλαγής οποιωνδήποτε ρυθμίσεων της ιδεατής μηχανής. Ο προγραμματιστής υλοποιεί το πρόγραμμά του χρησιμοποιώντας αυτές τις ρουτίνες σε C, C++, η FORTRAN κώδικα.

Παράλληλες Ιδεατές Μηχανές έχουν χρησιμοποιηθεί σε υλοποιήσεις αλγορίθμων και προβλημάτων βελτιστοποίησης. Μια παράλληλη υλοποίηση του αλγόριθμου Benders (φωλιασμένου) για μεγάλης κλίμακας γραμμικά προβλήματα που μοντελοποιούν δυναμικά στοχαστικά γραμμικά προβλήματα, υλοποιήθηκε σε δίκτυα απλών σταθμών εργασίας και σε IBM SP-2 (Dempster and Thompson 1997). Οι Linderoth et al (1999) περιγράφουν μια παράλληλη γραμμική ευριστική διαδικασία για την επίλυση προβλημάτων διαχωρισμού συνόλων η οποία υπολογίζει καλά αποτελέσματα σε μικρό χρόνο. Ο Zakarian (1995) παρουσιάζει μια μέθοδο για το πρόβλημα ροής σε δίκτυα πολλών προϊόντων μεγάλης κλίμακας και την παραλληλοποίησή της με τη χρήση cluster. Μια ενοποιημένη πλατφόρμα για την υλοποίηση παραλλήλων αλγορίθμων φραγμού και κλάδου έχει κατασκευαστεί στο εργαστήριο Prism (Benaïchouche et al 1996, Le Cun et al 1995).

5.4.2 MPI

Το MPI (*Message Passing Interface*) (Foster 1995; MPI Forum 1994; MPI Forum 1995; Gropp et al 1994; Gropp et al 1998) είναι μια πρόταση για τήν κανονικοποίηση του συστήματος ανταλλαγής μηνυμάτων για παράλληλες μηχανές κατανεμημένης μνήμης.

Ο σκοπός αυτού του προτύπου είναι η δημιουργία υπόβαθρου για την μεταφορά προγραμμάτων μεταξύ διαφορετικών παραλλήλων μηχανών. Απλώς καθορίζει ένα πρότυπο, όχι ένα πλήρες προγραμματιστικό περιβάλλον. Για αυτό το λόγο, δεν παρέχει ευκολίες σχεδιασμού προγραμμάτων και αποσφαλματοποίησης. Δεν παρέχει επίσης ορισμό διαχείρισης λαθών και προϋποθέτει ότι το προγραμματιστικό περιβάλλον είναι αξιόπιστο.

Το κύριο σώμα του MPI αποτελείται από ρουτίνες για επικοινωνία σημείου με σημείο. Αυτές οι ρουτίνες ενεργοποιούνται με δυο τρόπους: blocking και non-blocking. Τρεις τρόποι επικοινωνίας είναι διαθέσιμοι: ready (μια διαδικασία μπορεί να στείλει μήνυμα μόνο εάν η λαμβάνουσα έχει αρχικοποιηθεί); standard (ένα μήνυμα μπορεί να σταλεί ακόμη και εάν δεν υπάρχει ακόμη η λαμβάνουσα διαδικασία), και synchronous (όμοια με την προηγούμενη μόνο που η αποστέλλουσα διαδικασία μπορεί να τερματιστεί μόνο εάν η λαμβάνουσα έχει αρχικοποιηθεί). Η επιλογή του τρόπου επικοινωνίας εξαρτάται από τις ανάγκες κάθε εφαρμογής και μπορεί να επηρεάσει σημαντικά τη λειτουργία του προγράμματος. Οι ρουτίνες επικοινωνίας ομάδων συντονίζουν τις διαδικασίες που ανήκουν σε μια προκαθορισμένη ομάδα επεξεργαστών. Αυτές οι ρουτίνες επιτρέπουν την ομαδοποιημένη μετάδοση δεδομένων παρέχοντας τις παρακάτω λειτουργίες, Εικόνα 5.4.2.1: στέλνοντας τα ίδια δεδομένα από ένα μέλος της ομάδας σε όλα τα υπόλοιπα μέλη (broadcast); στέλνοντας διαφορετικά δεδομένα σε κάθε παραλήπτη (scatter); Στέλνοντας δεδομένα από όλα τα μέλη σε συγκεκριμένο παραλήπτη (gather); Στέλνοντας δεδομένα από όλα τα μέλη σε όλα τα μέλη (allgather); Και στέλνοντας διαφορετικά δεδομένα από όλα τα μέλη σε όλα τα μέλη (alltoall).

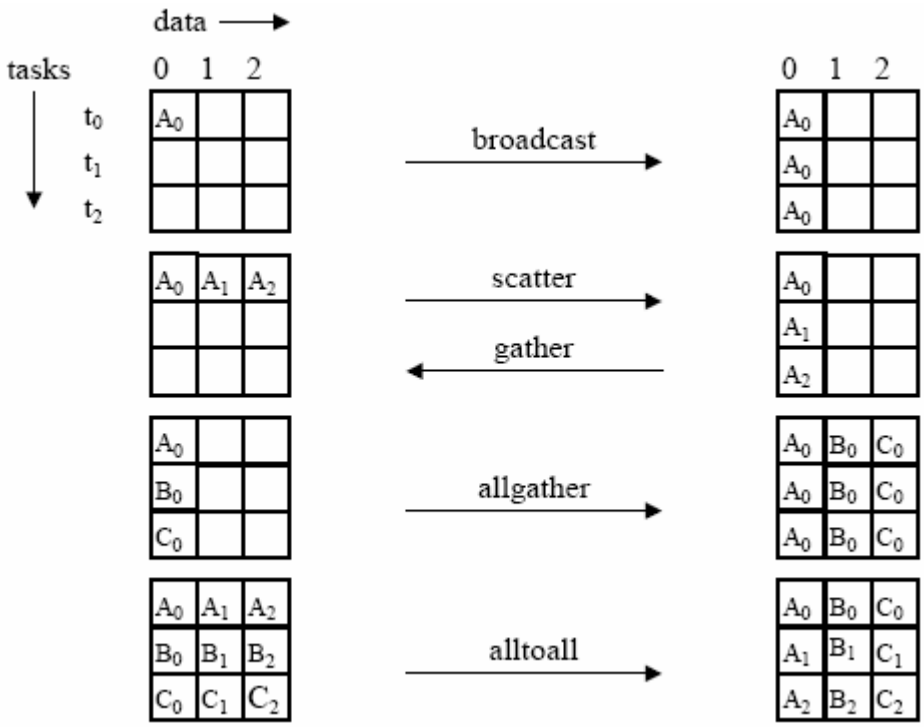


Figure 5.4.2.1: MPI routines for global communication

Υπάρχει επίσης μια διεργασία ελαχιστοποίησης για το υπολογισμό συναρτήσεων που αφορούν δεδομένα που ανήκουν σε μέλη μιας ομάδας επεξεργαστών. Η διαδικασία ελαχιστοποίησης εφαρμόζεται σε προκαθορισμένες ή ορισμένες από το χρήστη συναρτήσεις σε δεδομένα τοποθετημένα σε διαφορετικούς συμμετέχοντες επεξεργαστές. Το αποτέλεσμα κάθε τέτοιας συνάρτησης μπορεί να σταλεί σε όλα τα μέλη μιας ομάδας, ή σε ένα από αυτά. Ένα παράδειγμα είναι ο ορισμός ενός συνόλου αριθμών, στο οποίο κάθε αριθμός ανήκει στην περιοχή μνήμης διαφορετικών ομάδων.

Πολλές υλοποιήσεις του MPI είναι διαθέσιμες. Όλες είναι βασισμένες σε παράλληλες ιδεατές μηχανές, που απαρτίζονται από δια-συνδεδεμένους ετερογενείς υπολογιστές (workstations, PCs, multiprocessors) κάθε ένας από τους οποίους εκτελεί μια διεργασία για

τον έλεγχο ανταλλαγής μηνυμάτων. Ανάμεσα σ' αυτές τις υλοποιήσεις, η *mpich* έχει αναπτυχθεί από το Argonne National Laboratory και βασίζεται στο MPI 1.1 πρότυπο. Παρέχετα δωρεάν και είναι διαθέσιμο στη διεύθυνση http://www-unix.mcs.anl.gov/mpi/mpich. Αυτό το εργαλείο μπορεί να χρησιμοποιηθεί στις παρακάτω παράλληλες μηχανές: IBM SP-2, Intel Paragon, Cray T3D, Meiko CS-2, TMC CM-5, NCUBE NCUBE-2, Convex Exemplar, SGI Challenge, Sun Multiprocessors, CRAY YMP, και CRAY C-90. Επίσης τρέχει σε PCs (με Linux, FreeBSD, ή Windows), και Sun, HPO, SGI, IBM RS/6000, καθώς και DEC workstations. LAM ακόμη ένα δωρεάν πακέτο διαθέσιμο στη διεύθυνση http://www.mpi.nd.edu/lam, το οποίο βασίζεται στο MPI 2.0 πρότυπο και εχει αναπτυχθεί στο Laboratory for Scientific Computing του πανεπιστημίου της Notre Dame. Αυτή η υλοποίηση παρέχει ορισμένες ρουτίνες ελέγχου, οι οποίες είναι πάρα πολύ χρήσιμες. Έχει ήδη ελεγχθεί και πιστοποιηθεί για Solaris, IRIX, AIX, Linux, και HP-UX. Και τα δυο εργαλεία μπορούν να εγκατασταθούν και να χρησιμοποιηθούν με μεγάλη ευκολία.

Ορισμένες εταιρίες έχουν υλοποιήσει το πρότυπο για ειδικές περιπτώσεις παραλλήλων μηχανών Alpha, Cray, IBM, SGI, DEC, και HP. Το Cornell Theory Center παρέχει πολλές χρήσιμες πληροφορίες σχετικές με το MPI στη διεύθυνση http://www.tc.cornell.edu/UserDoc/SP/.

Υπάρχουν επίσης υλοποιήσεις αλγορίθμων για την επίλυση προβλημάτων βελτιστοποίησης χρησιμοποιώντας MPI. Μια βιβλιοθήκη σε C++ έχει κατασκευαστεί από τους Eckstein et al (1997) για κινητές, αρθρωτές, παράλληλες υλοποιήσεις αλγορίθμων φραγμού και διακλάδωσης. Το ZRAM (Marzetta et al 1999) είναι μια κινητή παράλληλη βιβλιοθήκη για αλγόριθμους αναζήτησης και έχει μοντελοποιηθεί πάνω σε ένα υποσύνολο του προτύπου ανταλλαγής μηνυμάτων του MPI. Οι Brüngger et al (1997) χρησιμοποιούν αυτή τη βιβλιοθήκη για την επίλυση οντοτήτων του τετραγωνικού προβλήματος ανάθεσης. Ο Homer (1997) περιγράφει την ανάπτυξη και υλοποίηση ενός παραλλήλου αλγορίθμου για την

προσέγγιση της μέγιστης τομής σε μη κατευθυνόμενους γράφους με βάρη. Ένα γενικό πλαίσιο για παράλληλους φραγμού και διακλάδωσεις αλγορίθμους με υποστήριξη δυναμικής κατανομής φόρτου εργασίας υπάρχει στους (Batoukov and Serevik 1999).

Πολλοί παράλληλοι αλγόριθμοι χρησιμοποιούν μετα-ευρεστικές μεθόδους με τη βοήθεια του MPI. Οι Martins et al (1998, 1999) ανέπτυξαν παράλληλες υλοποιήσεις για άπληστες προσαρμοζόμενες στοχαστικές διαδικασίες αναζήτησεις (GRASP, see e.g. (Feo and Resende 1995)) για την επίλυση του προβλήματος του Steiner σε γράφους. Μια γενική αντικειμενοστρεφής βιβλιοθήκη σε C++ και MPI, υλοποιήθηκε από τους Kliewer και Tschöke (1998), και παρέχει ένα πλαισιο για προσομοίωση ισχυρών συστημάτων βελτιστοποίησης πράγμα που μπορεί να εφαρμοστεί σε πολλά προβλήματα βελτιστοποίησης. Οι Fachat και Hoffmann (1997) περιγράφουν την ανάπτυξη μιας εφαρμογής βασισμένη στη μιμούμενη ανόπτηση με τη δυναμική εξισορρόπηση φορτίων για να πάρουν τη μέγιστη χρήση της διαθέσιμης δύναμης επεξεργασίας. Ο Salhi (1998) περιγράφει μια παράλληλη υλοποίηση ενός εργαλείου για παράλληλη παλινδρόμηση, βασιζόμενος σε γενετικούς αλγορίθμους. Οι Rivera-Gallego (1998) παρουσιάζουν έναν γενικό παράλληλο γενετικό αλγόριθμο για την τρισδιάστατη απεικόνιση σημείων για τη δημιουργία Ευκλειδείων πινάκων αποστάσεων και την παραλληλοποίησή τους.

5.5 Εργαλεία αξιολόγησης απόδοσης

Είναι πολύ σημαντικό να χρησιμοποιηθούν εμπειρικά στοιχεία προκειμένου να αξιολογηθεί η απόδοση των παράλληλων εφαρμογών. Το πρώτο βήμα προς τη βελτίωση εκτέλεσης συνίσταται στη συλλογή των κατάλληλων και αξιόπιστων στοιχείων εφαρμογής (Foster 1995). Υπάρχουν τρεις βασικές κατηγορίες συλλεκτών στοιχείων: profilers, μετρητές, και ίχνη γεγονότος.

Τα Profilers καταγράφουν το χρονικό διάστημα που ξοδεύεται στα διαφορετικά μέρη ενός προγράμματος, συνήθως στις υπορουτίνες. Αυτές οι πληροφορίες λαμβάνονται συχνά με τη δειγματοληψία του μετρητή προγράμματος και την παραγωγή ενός ιστογράμματος των συχνοτήτων εκτέλεσης. Αυτές οι συχνότητες συνδυάζονται με τις επιτραπέζιες πληροφορίες συμβόλων μεταγλωττιστών για να υπολογίσουν το χρονικό διάστημα που ξοδεύεται στα διαφορετικά τμήματα προγράμματος. Τα περισσότερα εργαλεία Profilers παρέχονται από τον προμηθευτή και είναι κατάλληλα για συγκεκριμένους Η/Υ, αναλόγως. Συνήθως, ο χρήστης διευκρινίζει μια επιλογή για το μεταγλωττιστή να παράγει τα στοιχεία για το profiling κατά τη διάρκεια της εκτέλεσης του προγράμματος. Ένα παράλληλο profiler συνήθως συγκεντρώνει και δείχνει τις πληροφορίες από κάθε στόχο του παράλληλου προγράμματος. Τα Profilers μπορεί να είναι πολύ χρήσιμα, στον προσδιορισμό εκείνου του μέρους ενός προγράμματος που καταναλώνει τα μεγαλύτερα χρονικά διαστήματα, επιτρέποντας να γίνει καλύτερη βελτιστοποίηση του προγράμματος. Μπορούν επίσης να προσδιορίσουν τα τμήματα προγράμματος που δεν αυξάνουν ορθώς βαθμιδωτά σε σχέση με το μέγεθος εισαγωγής ή με τον αριθμό επεξεργαστών.

Οι μετρητές καταγράφουν κάθε φορά που εμφανίζεται ένα συγκεκριμένο γεγονός και μπορεί να απαιτήσουν κάποιο πρόσθετο προγραμματισμό από τον προγραμματιστή. Μπορούν να χρησιμοποιηθούν για να καταγράψουν τα σφαιρικά γεγονότα του προγράμματος, όπως ο αριθμός κλήσεων διαδικασιών, ο συνολικός αριθμός μηνυμάτων, και ο αριθμός μηνυμάτων που ανταλλάσσονται μεταξύ των ζευγαριών των επεξεργαστών, τα οποία δεν μπορούν να ληφθούν από τα profilers.

Τα ίχνη γεγονότος καταγράφουν κάθε περιστατικό των συγκεκριμένων γεγονότων (όπως οι κλήσεις διαδικασιών ή τα μηνύματα που ανταλλάσσονται) και παράγουν τα αρχεία ημερολογίου που περιέχουν αυτά τα αρχεία. Μπορούν να χρησιμοποιηθούν για να ερευνήσουν τις σχέσεις μεταξύ των επικοινωνιών και να καθορίσουν τις πηγές αδρανούς

χρόνου και δυσχερειών επικοινωνίας. Αυτή η προσέγγιση παράγει έναν τεράστιο όγκο των στοιχείων, ο οποίος μπορεί να είναι δύσκολο να χειριστεί και να αποθηκευθεί. Η καταγραφή αυτού του τύπου στοιχείων τείνει να αλλάξει τα χαρακτηριστικά προγράμματος, που καλύπτουν τις αρχικές πηγές προβλημάτων (έλεγχος-επίδραση). Η μετά-επεξεργασία είναι δύσκολο να εκτελεσθεί, λόγω του μεγάλου ποσού δεδομένων, και ο προγραμματιστής μπορεί να πρέπει να ξοδέψει κάποια προσπάθεια ώστε να καταγραφούν μόνο τα σχετικά γεγονότα. Αυτή η τεχνική απαιτεί συνήθως κάποιο πρόσθετο προγραμματισμό από τον προγραμματιστή.

Το ακατέργαστο στοιχείο που παράγεται από τέτοιους συλλέκτες στοιχείων μετασχηματίζεται συνήθως για να μειώσει τα συνολικά αποθηκευμένα στοιχεία, ώστε να ερμηνευθεί ευκολότερα και να εκτεθεί από τα εργαλεία απεικόνισης. Τα ιστογράμματα χρησιμοποιούνται συχνά στην παρουσίαση στοιχείων που συλλέγονται από τα profilers, τους μετρητές, και τα ίχνη γεγονότος. Μπορούν να επιδείξουν πληροφορίες όπως το μέσο μέγεθος μηνυμάτων, ο συνολικός χρόνος υπολογισμού, ή ο όγκος επικοινωνίας, ως λειτουργία του χρόνου. Για τα ίχνη γεγονότος, μερικές μορφές έκθεσης μπορούν να παρέχουν τις λεπτομερείς απόψεις των προσωρινών σχέσεων μεταξύ των διαφορετικών επεξεργαστών. Παραδείγματος χάριν, διαγράμματα όπου κάθε στήλη αντιπροσωπεύει την κατάσταση ενός επεξεργαστή (εάν κάνει υπολογισμούς, εάν επικοινωνεί, ή εάν δεν είναι απασχολημένος) ως λειτουργία του χρόνου. Στο υπόλοιπο αυτού του τμήματος, παρουσιάζουμε μερικά ελεύθερα προσβάσιμα καθώς και εμπορικά πακέτα για την ανάλυση απόδοσης των εργαλείων λογισμικού που παρουσιάζονται στο προηγούμενο τμήμα.

Το Gprof είναι ένα εμπορικό εργαλείο που παρέχεται από την IBM για να κάνει profiling τα παράλληλα προγράμματα που αναπτύσσονται χρησιμοποιώντας τη βιβλιοθήκη MPI της IBM. Παρουσιάζει χρόνους εκτέλεσης των υπορουτινών που εκτελούνται από κάθε επεξεργαστή. Εκτός από το gprof, όλα τα άλλα εργαλεία που περιγράφονται σε αυτό το τμήμα είναι βασισμένα στα ίχνη γεγονότος.

Το Nupshot είναι ένα εργαλείο απόδοσης που διανέμεται μαζί με την mpich υλοποίηση της MPI του Εθνικού εργαστηριακού Argonne. Ο χρήστης εισάγει κλήσεις στις ρουτίνες καταγραφών στον πηγαίο κώδικα, μεταγλωττίζει και κάνει τη σύνδεση με τη βιβλιοθήκη profiling, αποκαλούμενη MPE MPI. Όταν ο κώδικας εκτελείται, ένα αρχείο ημερολογίου από χρονικά καταγεγραμμένα γεγονότα παράγεται. Αυτό το αρχείο ημερολογίου μπορεί να απεικονιστεί με το nupshot, που επιδεικνύει την κατάσταση των διαδικασιών κατά τη διάρκεια της εκτέλεσης του προγράμματος.

AIMS (Automated Instrumentation and Monitoring System) (Yan et al 1995, Yan and Sarukkai 1996) ένα ελεύθερα προσβάσιμο πακέτο που μπορεί να μεταφορτωθεί από την ιστοσελίδα http://science.nas.nasa.gov/Software/AIMS/ για τη μέτρηση και την ανάλυση των προγραμμάτων ανταλλαγής μηνυμάτων, σε FORTRAN 77 και C, χρησιμοποιώντας βιβλιοθήκες επικοινωνίας όπως MPI και PVM.

Το Paradyn (Miller et al 1995) είναι ένα εργαλείο για τη διαδραστική ανάλυση, κατά το χρόνο εκτέλεσης, παράλληλων προγραμμάτων, που χρησιμοποιούν το μήνυμα που περνά τις βιβλιοθήκες όπως PVM και MPI. Μπορεί να μεταφορτωθεί από την ιστοσελίδα http://www.cs.wisc.edu/paradyn/. Πριν τρέξει το πρόγραμμα, ο χρήστης καθορίζει τις απεικονίσεις απόδοσης, από την άποψη των μετρικά – εστιασμένων ζευγαριών. Οι μετρικές μπορούν να είναι, παραδείγματος χάριν, χρόνος επεξεργασίας, χρόνος φραξίματος, ή ποσοστά μηνυμάτων. Η εστίαση είναι ένας κατάλογος τμημάτων προγράμματος, όπως οι διαδικασίες, οι επεξεργαστές, ή τα κανάλια μηνυμάτων. Παραδείγματος χάριν, ο χρήστης μπορεί να διευκρινίσει μια χρονική απεικόνιση με μία μετρική του χρόνου επεξεργασίας και να εστιάσει σε ένα ιδιαίτερο υποσύνολο των διαδικασιών.

Το περιβάλλον ανάλυσης απόδοσης Pablo (DeRose et al 1998; Reed et al 1993) διαθέσιμο από την ιστοσελίδα http://vibes.cs.uiuc.edu/ είναι ένα ελεύθερα προσβάσιμο πακέτο που αποτελείται από διάφορα συστατικά για την ενοργάνωση και ανίχνευση

παράλληλων προγραμμάτων MPI, καθώς επίσης και για την ανάλυση των αρχείων ιχνών που παράγονται από τα ενοργανωμένα εκτελέσιμα.

Το VAMPIR (Visualization and analysis of MPI resource) (Galarowicz and Mohr 1998; agel et al 1996) είναι ένα εμπορικό εργαλείο απεικόνισης ιχνών από την Pallas GmbH διαθέσιμο για πολλές διαφορετικές πλατφόρμες. Ο προγραμματιστής ενοργανώνει τον κώδικα και τον συνδέει με τη βιβλιοθήκη VAMPIRtrace. Τα αρχεία ιχνών που παράγονται με τις διαδικασίες MPI κατά την εκτέλεση του προγράμματος συλλέγονται αυτόματα και συγχωνεύονται σε ένα ενιαίο αρχείο ιχνών. Τα τελευταίο μπορεί να το δει κανείς χρησιμοποιώντας το VAMPIR με την παρουσίαση απεικόνισης, που διαμορφώνεται από το χρήστη. Μπορεί επίσης να χρησιμοποιηθεί για να αναλύσει τα προγράμματα που εκτελούνται με HPF, μεταφρασμένα σε ένα πρόγραμμα MPI. Οι πληροφορίες για αυτό το εργαλείο είναι διαθέσιμες στην ιστοσελίδα http://www.pallas.de/pages/vampir.htm.

Το VT (Visualization Tool) μπορεί να χρησιμοποιηθεί για τη post – mortem απεικόνιση ιχνών ή για την σε απευθείας σύνδεση παρακολούθηση απόδοσης των παράλληλων προγραμμάτων που αναπτύσσονται με την υλοποίηση MPI της IBM. Είναι ένα εμπορικό εργαλείο που παρέχεται από την IBM και μπορεί μόνο να χρησιμοποιηθεί στο παράλληλο περιβάλλον SP της IBM.

Το XPVM είναι ένα ελεύθερα προσβάσιμο πακέτο διαθέσιμο από την ιστοσελίδα http://www.netlib.org/pvm3/ xpvm και παρέχει μια γραφική κονσόλα και ένα όργανο ελέγχου για το PVM. Ο χρήστης αρχίζει ένα πρόγραμμα PVM χρησιμοποιώντας XPVM και κάθε spawned γεγονός επιστρέφει πληροφορία ιχνών στόχου για την ανάλυση σε πραγματικό χρόνο ή για την post – mortem αναπαραγωγή από αποθηκευμένα αρχεία ιχνών.

5.6 Αποσφαλμάτωση ενός παράλληλου προγράμματος

Στον παράλληλο προγραμματισμό, τα ζητήματα που αφορούν την ταύτιση και διανομή οδηγούν στον μη-ντετερμινισμό: επαναλαμβανόμενες εκτελέσεις του ίδιου προγράμματος, με τα ίδια δεδομένα εισόδου, μπορούν να παρουσιάσουν διαφορετικές συμπεριφορές. Αυτό μπορεί να κάνει το έργο της αποσφαλμάτωσης πολύ δύσκολο. Μερικοί συντάκτες ακόμη προτείνουν ότι ο προγραμματιστής πρέπει να αποφύγει τον μη- ντετερμινισμό όποτε είναι δυνατόν (παραδείγματος χάριν, με τον περιορισμό του μηνύματος που προέρχεται κατά την εκτέλεση λήψης δηλώσεων) (Foster 1995). Αυτό, εντούτοις, μπορεί να οδηγήσει στην υπο-εκμετάλλευση του πιθανού παραλληλισμού της εφαρμογής.

Ένας κοινός, απλός τρόπος να αποσφαλματωθεί ένα παράλληλο πρόγραμμα είναι με την εισαγωγή των οδηγιών στον κώδικα ώστε να εκτυπωθούν οι σχετικές πληροφορίες. Αν και χρήσιμο, τα δεδομένα εξόδου στερούνται τις προσωρινές πληροφορίες, δεδομένου ότι οι πληροφορίες δεν είναι απαραιτήτως τυπωμένες στην ίδια σειρά όπως παράγονται από το παράλληλο πρόγραμμα, λόγω των στοιχείων buffering από τους επεξεργαστές.

Το PVM και το ελεύθερα προσβάσιμο πακέτο LAM MPI επιτρέπουν στον προγραμματιστή να εκκινήσει αποσφαλματωτές σε απόμακρυσμένους κόμβους. Οι στόχοι μπορούν έπειτα να εκτελεσθούν, υπό τον έλεγχο του αποσφαλματωτή, που καθιστά ευκολότερη την παρακουλουθήση εκτέλεσης του προγράμματος. Εντούτοις, αυτό το είδος τεχνικής μπορεί να συγκρουστεί με την εκτέλεση του παράλληλου προγράμματος, κάνοντάς το να συμπεριφερθεί διαφορετικά από τον κανονικό τρόπο που τρέχει. Το XPVM παρέχει επίσης κάποια απλή βοήθεια αποσφαλμάτωσης στους προγραμματιστές PVM, με την απαρίθμηση λεπτομερειών σε μορφή κειμένου από τα γεγονότα ιχνών και τη εξ' αποστάσεως ολοκλήρωση του στόχου.

Το Total View είναι ένας εμπορικός αποσφαλματωτής που αναπτύχθηκε από τους Etnus Inc. (Etnus 1999) και μπορεί να χρησιμοποιηθεί για να αποσφαλματώσει MPI, PVM, HPF, και πολύ – νηματικά προγράμματα. Επιτρέπει στον προγραμματιστή να θέσει σημεία παύσης και σημεία δράσης μέσα στα παράλληλα προγράμματα για να ελεγχθεί η εκτέλεσή τους. Παρέχει επίσης την απεικόνιση δεδομένων για τις μεταβλητές που συνδέονται με τα νήματα ή τις διαδικασίες, καθώς επίσης και με τους κατανεμημένους πίνακες HPF. Αν και χρήσιμο για την αποσφαλμάτωση παράλληλων προγραμμάτων, μπορεί να απαιτήσει κάποια προσπάθεια εκμάθησης από τον προγραμματιστή πριν από τη χρήση του.

5.7 Βιβλιοθήκες παράλληλης βελτιστοποίησης

Οι βιβλιοθήκες βελτιστοποίησης αποτελούνται από ένα σύνολο υπορουτινών που μπορεί να χρησιμοποιηθούν από τους προγραμματιστές εφαρμογών για να λύσουν μεγάλα προβλήματα βελτιστοποίησης. Περιλαμβάνουν συναρτήσεις για να λύσουν τα προβλήματα γραμμικού και μικτού – ακέραιου προγραμματισμού και παρέχουν μερικά χαρακτηριστικά γνωρίσματα, όπως οι μεταβλητές ελέγχου (ρουτίνες για να επιτρέψουν στους χρήστες να αλλάξουν μια ποικιλία προεπιλογμένων ρυθμίσεων παραμέτρων που ελέγχουν τους αλγορίθμους), ικανότητες αποσφαλμάτωσης (ρουτίνες για να παρέχουν στατιστικές για τις τιμές στοιχείων, όπως οι λογικές στόχου και περιορισμών, τα όρια και οι συντελεστές στη μήτρα περιορισμών και αποθήκευση που χρησιμοποιείται κατά τη διάρκεια της εκτέλεσης) και ανάγνωση – γράψιμο αρχείων (ρουτίνες που διαβάζουν σε διάφορες μορφές εισαγωγής και παράγουν τα αρχεία λύσης εξόδου). Δύο ευρέως γνωστά εμπορικά συστήματα βελτιστοποίησης, τα CPLEX και OSL, παρέχουν τις παράλληλες εφαρμογές των σειριακών βιβλιοθηκών βελτιστοποίησής τους. Το CPLEX (CPLEX 1999) είναι ένα καταχωρημένο εμπορικό σήμα από των ILOG. Παρέχει μια βιβλιοθήκη βελτιστοποίησης για τα συστήματα Unix και για PCs που τρέχουν το DOS, τα Windows 3,1, τα WINDOWS NT, τα

WINDOWS.95, και OS/2. Οι παράλληλοι λύτες CPLEX εργάζονται στις παράλληλες αρχιτεκτονικές πολυεπεξεργαστών κατανεμημένης μνήμης, όπως οι DEC, Hewlett - Packard, Silicon Graphics, Sun και βασισμένων στην Intel εκδόσεις υπολογιστών για τους οποίους η παράλληλη υποστήριξη CPLEX έχει αναπτυχθεί. Οι παράλληλες εκδόσεις Barrier solver και μικτού ακεραίου λύτη ακέραιων αριθμών είναι διαθέσιμες για όλες τις παράλληλες πλατφόρμες. Η διαθεσιμότητα του παράλληλου αλγορίθμου Simplex ποικίλλει με την πλατφόρμα. Ο χρήστης επιλέγει το βαθμό παραλληλισμού που θα χρησιμοποιηθεί πριν αρχίσει μια βελτιστοποίηση και το CPLEX εφαρμόζει αυτόματα την παραλληλοποίηση.

Το OSLp, η βιβλιοθήκη παράλληλων υπορουτινών βελτιστοποίησης της IBM (IBMa 1999), περιλαμβάνει όλες τις συναρτήσεις της σειριακής βιβλιοθήκης OSL. Το OSLp τρέχει στην SP της IBM RS/6000 στα βαθμιδωτά συστήματα POWER parallel και στις συστάδες των τερματικών σταθμών RS/6000s. Το OSLp τρέχει στο πλαίσιο του παράλληλου περιβάλλοντος AIX. Το OSLp παρέχει ένα σύνολο πάνω από 60 υπορουτινών καλέσιμων από τα προγράμματα εφαρμογών που γράφονται σε XL FORTRAN ή C, για να λύσουν προβλήματα γραμμικού προγραμματισμού, μικτού-ακέραιου προγραμματισμού, τετραγωνικού προγραμματισμού και προβλήματα βελτιστοποίησης. Μερικές από αυτές τις υπορουτίνες χρησιμοποιούν σειριακούς αλγορίθμους, ενώ άλλοι χρησιμοποιούν τους αλγορίθμους που εκμεταλλεύονται τις ικανότητες του παράλληλου περιβάλλοντος, έτσι ώστε πολλαπλάσιοι επεξεργαστές μπορούν ταυτόχρονα να εκτελέσουν υπολογισμούς στις δευτερεύουσες υποχρεώσεις του προβλήματος που πρέπει να επιλυθεί. Μόνο ελάχιστες αλλαγές στις σειριακές εφαρμογές OSL απαιτούνται για να παραγάγουν προγράμματα παράλληλων εφαρμογών και κανένας ειδικός παράλληλος προγραμματισμός δεν απαιτείται. Περισσότερες πληροφορίες για αυτήν την βιβλιοθήκη μπορούν να βρεθούν στην ιστοσελίδα http://www.rs6000.ibm.com/software/sp-products/oslp.html.

5.8 Μετρικές απόδοσης για τα παράλληλα συστήματα

Ο κύριος στόχος της γραφής ενός παράλληλου προγράμματος είναι να αποκτηθεί ένας πιο σύντομος χρόνος υπολογισμού έναντι της σειριακής έκδοσης. Έχοντας αυτό υπόψη, υπάρχουν διάφορα ζητήματα που πρέπει να θεωρήσουμε κατά τη σχεδιασμό του παράλληλου κώδικά μας για να λάβουμε την καλύτερη δυνατή απόδοση, στα πλαίσια των περιορισμών της επίλυσης του προβλήματος. Σε αυτήν την ενότητα, εισάγουμε μερικές μετρικές που χρησιμοποιούνται συνήθως για να μετρήσουν την απόδοση των παράλληλων συστημάτων.

i) Εξισορρόπηση φορτίων

Η εξισορρόπηση φορτίων είναι ο στόχος κατά τον οποίο εξισώνεται η εργασία μεταξύ των διαθέσιμων διαδικασιών. Αυτό μπορεί να γίνει εύκολα όταν εκτελούνται οι ίδιες ενέργειες από όλες τις διαδικασίες (σε διαφορετικά τμήματα δεδομένων). Δεν είναι τετριμμένο όταν εξαρτάται ο χρόνος επεξεργασίας από τις τιμές στοιχείων που επεξεργάζονται. Όταν υπάρχουν μεγάλες αποκλίσεις στο χρόνο επεξεργασίας, μπορεί να απαιτείται να υιοθετήσουμε μια διαφορετική μετρική.

ii) Ελαχιστοποίηση της επικοινωνίας

Ο συνολικός χρόνος εκτέλεσης είναι μια σημαντική ανησυχία στον παράλληλο προγραμματισμό επειδή είναι ένα ουσιαστικό συστατικό για τη σύγκριση και βελτίωση όλων των προγραμμάτων.

Τρίς συνιστώσεις συμβάλλουν στο χρόνο εκτέλεσης:

1- Ο χρόνος υπολογισμού είναι ο χρόνος που ξοδεύεται για την εκτέλεση των υπολογισμών στα δεδομένα. Ιδανικά, πρέπει να αναμένουμε ότι εάν είχαμε n επεξεργαστές που εργάζονται σε ένα πρόβλημα, πρέπει να είμαστε σε θέση να τελειώσουμε την εργασία στο 1/n ποσοστό του χρόνου της σειριακής εργασίας. Αυτό θα συνέβαινε εάν ο χρόνος όλου του επεξεργαστή ξοδεύονταν στον υπολογισμό.

2- Ο χρόνος αδράνειας είναι ο χρόνος που μια διαδικασία δαπανά για την αναμονή κάποιων δεδομένων από άλλους επεξεργαστές. Κατά τη διάρκεια αυτής της περιόδου, οι επεξεργαστές δεν κάνουν καμία χρήσιμη εργασία.

3- Ο χρόνος επικοινωνίας είναι ο χρόνος που απαιτείται για τις διαδικασίες αποστολής και λήψης των μηνυμάτων. Το κόστος της επικοινωνίας στο χρόνο εκτέλεσης μπορεί να μετρηθεί από την άποψη της λανθάνουσας κατάστασης (latency) και του εύρους ζώνης (bandwidth). Η λανθάνουσα κατάσταση είναι ο χρόνος που απαιτείται για να οργανώσει το φάκελο για την επικοινωνία, όπου το εύρος ζώνης είναι η πραγματική ταχύτητα της μετάδοσης, ή bits ανά μονάδα χρόνου. Τα σειριακά προγράμματα δεν χρησιμοποιούν διαδικαστική (interprocess) επικοινωνία. Επομένως, πρέπει να ελαχιστοποιήσουμε αυτήν την χρήση του χρόνου, ώστε να πάρουμε τις καλύτερες βελτιώσεις απόδοσης.

iii) Επικαλύπτοντας επικοινωνία και υπολογισμό.

Υπάρχουν διάφοροι τρόποι να ελαχιστοποιηθεί ο χρόνος μη απασχόλησης μεταξύ των διαδικασιών, και ένα παράδειγμα είναι η επικάλυψη της επικοινωνίας και του υπολογισμού. Αυτό περιλαμβάνει την κατάληψη μιας διαδικασίας με έναν ή περισσότερους νέους στόχους ενώ περιμένει την επικοινωνία να τελειώσει έτσι ώστε να μπορεί να προχωρήσει σε έναν άλλο στόχο. Η προσεκτική χρήση της non-blocking επικοινωνίας και ο μη συγκεκριμένος υπολογισμός δεδομένων καθιστούν αυτό πιθανό. Είναι πολύ δύσκολο στην πράξη να παρεμβληθεί η επικοινωνία με τον υπολογισμό.

iv) Χρόνος εκτέλεσης

Από τη στιγμή όπου οι υπολογισμοί επιτάχυνσης εμφανίζονται να είναι ο κύριος λόγος πίσω από το ενδιαφέρον μας για την οικοδόμηση των παράλληλων υπολογιστών, το σημαντικότερο μέτρο στην αξιολόγηση ενός παράλληλου αλγορίθμου είναι επομένως ο

χρόνος εκτέλεσης του. Αυτό ορίζεται ως ο χρόνος που λαμβάνεται από τη στιγμή της έναρξης του αλγορίθμου μέχρι τη στιγμή που ολοκληρώνεται. Εάν οι διάφοροι επεξεργαστές δεν αρχίζουν όλοι και δεν τελειώνουν τον υπολογισμό τους ταυτόχρονα, ο χρόνος εκτέλεσης είναι ίσος με το χρόνο που παρήλθε μεταξύ της στιγμής που ο πρώτος επεξεργαστής στον παράλληλο υπολογιστή άρχισε να δουλεύει και τη στιγμή που ο τελευταίος επεξεργαστής παράγει τα δεδομένα εξόδου ολοκληρώνοντας.

Ο χρόνος εκτέλεσης ενός παράλληλου αλγορίθμου λαμβάνεται συνήθως με τον υπολογισμό δύο ειδών βημάτων, υπολογιστικά βήματα και βήματα δρομολόγησης. Κάθε ένα από αυτά τα βήματα απαιτεί έναν σταθερό αριθμό χρονικών μονάδων: Ένα υπολογιστικό βήμα είναι μια αριθμητική ή λογική λειτουργία που εκτελείται σε ένα στοιχείο (datum) μέσα σε έναν επεξεργαστή. Σε ένα βήμα δρομολόγησης, ένα στοιχείο (datum) ταξιδεύει από έναν επεξεργαστή σε άλλο μέσω της μοιραζόμενης μνήμης ή μέσω του δικτύου επικοινωνίας.

Για ένα πρόβλημα του μεγέθους n ο χρόνος της χειρότερης περίπτωσης εκτέλεσης ενός παράλληλου αλγορίθμου, μια συνάρτηση του n, θα συμβολίζεται με $t_p(n)$.

v) Επιτάχυνση Sp.

Στην αξιολόγηση ενός παράλληλου αλγορίθμου για ένα δεδομένο πρόβλημα, είναι αρκετά φυσικό να γίνει από την άποψη του καλύτερου διαθέσιμου σειριακού αλγορίθμου για εκείνο το πρόβλημα. Κατά συνέπεια μια καλή ένδειξη της ποιότητας ενός παράλληλου αλγορίθμου είναι η επιτάχυνση που παράγει. Αυτό ορίζεται ως

$$Sp = t_{\text{seq}}(n) / t_p(n)$$

όπου $t_{seq}(n)$ δείχνει το χρόνο εκτέλεσης στη χειρότερη περίπτωση του γρηγορότερα γνωστού σειριακού αλγορίθμου για το δεδομένο πρόβλημα. Σαφώς, όσο μεγαλύτερη η επιτάχυνση, τόσο καλύτερος ο παράλληλος αλγόριθμος.

vi) Αριθμός επεξεργαστών

Ένα από τα σημαντικότερα κριτήρια στην αξιολόγηση ενός παράλληλου αλγορίθμου είναι ο αριθμός επεξεργαστών που απαιτείται για να λυθεί ένα πρόβλημα. Κοστίζει να αγοράσεις, διατηρήσεις, και να λειτουργήσεις υπολογιστές. Επομένως, όσο μεγαλύτερος ο αριθμός επεξεργαστών που χρησιμοποιούνται από έναν αλγόριθμο για να λύσει ένα πρόβλημα, τόσο ακριβότερη γίνεται η λύση. Δηλώνουμε τον αριθμό επεξεργαστών που απαιτούνται από έναν αλγόριθμο για να λύσουν ένα πρόβλημα του μεγέθους n με $p(n)$. Μερικές φορές ο αριθμός επεξεργαστών είναι μια σταθερά ανεξάρτητη από n.

vii) Το κόστος C(n).

Το κόστος $C(n)$ από έναν παράλληλο αλγόριθμο ορίζεται ως το γινόμενο του χρόνου εκτέλεσης ενός παράλληλου αλγορίθμου και του αριθμού επεξεργαστών, ως εκ τούτου $C(n) = t_p(n) \times p(n)$ Το κόστος ενός παράλληλου αλγορίθμου για ένα δεδομένο πρόβλημα λέγεται ότι είναι βέλτιστου κόστους, εάν το κόστος του παράλληλου αλγορίθμου ταιριάζει με το κάτω όριο του αριθμού σειριακών διαδικασιών που απαιτούνται στη χειρότερη περίπτωση για να λύσουν το πρόβλημα. Ένας παράλληλος αλγόριθμος δεν είναι βέλτιστου κόστους, εάν ένας σειριακός αλγόριθμος υπάρχει, του οποίου ο χρόνος εκτέλεσης είναι μικρότερος από το κόστος του παράλληλου αλγορίθμου.

viii) Η αποδοτικότητα E(n).

Η αποδοτικότητα ενός παράλληλου αλγορίθμου που τρέχει *σε p* επεξεργαστές είναι η επιτάχυνση διαιρούμενη με p. Ας δώσουμε το ακόλουθο παράδειγμα που επεξηγεί την έννοια του αποδοτικού παράλληλου αλγορίθμου. Εάν ο καλύτερα γνωστός σειριακός αλγόριθμος εκτελείται σε 12 δευτερόλεπτα (σε έναν επεξεργαστή), ενώ ένας παράλληλος αλγόριθμος που λύνει το ίδιο πρόβλημα εκτελείται σε 3 δευτερόλεπτα όταν χρησιμοποιούνται 5 επεξεργαστές, θα λέμε ότι ο παράλληλος αλγόριθμος παρουσιάζει επιτάχυνση $S_5 = 4$ με πέντε επεξεργαστές.

Ένας παράλληλος αλγόριθμος που παρουσιάζει μια επιτάχυνση 4 με πέντε επεξεργαστές έχει μια αποδοτικότητα 0,8 με πέντε επεξεργαστές. Η αποδοτικότητα $E(n)$ ενός παράλληλου αλγορίθμου για την επίλυση ενός προβλήματος καθορίζεται ως εξής:

$$E(n) = t_{seq}(n)/C(n).$$

5.9 Κίνητρο για μια σειριακή και κατανεμημένη υλοποίησης πλήρους ταμπλώ.

5.9.1 Γιατί η μέθοδος εφαρμόζεται στο πλήρες ταμπλώ

Στην υπολογιστική μας μελέτη χρησιμοποιούμε την μέθοδο πλήρους ταμπλώ. Υπάρχουν διάφοροι λόγοι για αυτό. Οι δύο κύριοι λόγοι είναι ότι είναι καλύτερη για τα πυκνά προβλήματα και ότι ένα ακατέργαστο κατανεμημένο πρόγραμμα είναι απλό. Πιο συγκεκριμένα:

5.9.1.1 Καμία έξω τιμολόγηση ή αναδημιουργία στηλών

Όπως αναφέρθηκε προηγουμένως η αναθεωρημένη μέθοδος απαιτεί πρόσθετο υπολογισμό για να υπολογίσει την αντικειμενική γραμμή. Απαιτεί επίσης πρόσθετο υπολογισμό για να υπολογίσει τη στήλη που εισάγεται στη βάση. Η χρησιμοποίηση της μεθόδου πλήρους ταμπλώ αποφεύγει αυτούς τους υπολογισμούς.

5.9.1.2 Εναλλακτικοί κανόνες επιλογής στηλών μπορούν εύκολα να χρησιμοποιηθούν

Ένα άλλο πλεονέκτημα την μέθοδο πλήρους ταμπλό είναι η δυνατότητα χρήσης πολλαπλάσιων κανόνων επιλογής στηλών. Ο κλασσικός κανόνας (Dantzig) επιλέγει απλά τη στήλη με το μέγιστο συντελεστή στην αντικειμενική γραμμή του ανανεωμένου πλήρους ταμπλώ. Είναι εύκολο να χρησιμοποιηθεί αυτός ο κανόνας. Το κόστος επιλογής της στήλης που θα εισαχθεί στη βάση, που χρησιμοποιεί ο κανόνας Dantzig, είναι μόνο n συγκρίσεις. Η

αναθεωρημένη μέθοδος όπως αναφέρεται στις προηγούμενες ενότητες πρέπει να υπολογίσει από την αρχή τα ελλειματικά κόστη, μια διαδικασία που εκτελείται με m^2+mn πράξεις.

5.9.1.3 Η αναθεωρημένη μορφή δεν πλεονεκτεί στα πυκνά προβλήματα

Η μέθοδός μας λειτουργεί για όλα τα προβλήματα, αλλά αποδίδει καλύτερα όταν χρησιμοποιείται σε πυκνά προβλήματα. Αυτό είναι επειδή η αναθεωρημένη μέθοδος είναι πιο αργή από την μέθοδο πλήρους ταμπλώ για πυκνά προβλήματα ακόμη και σε έναν επεξεργαστή, υποθέτοντας ότι το *n* δεν είναι πάρα πολύ μεγαλύτερος από το *m*. Για τα πυκνά προβλήματα δεν υπάρχει λόγος χρησιμοποίησης της αναθεωρημένης μεθόδου. Εφαρμογές στις οποίες συναντώνται πυκνά προβλήματα δίνονται στην επόμενη ενότητα.

5.10 Εφαρμογές με πυκνές μήτρες

Υπάρχουν διάφορες εφαρμογές που οδηγούν σε πυκνά γραμμικά προγράμματα. Μια απ`αυτές είναι η εξώρυξη δεδομένων [Bradley, 1999] και η κατηγοριοποίηση κειμένων που χρησιμοποιεί τη μέθοδο των [Bennett and Mangasarian, 1992], και των [Bosch and Smith, 1998]. Η ιδέα είναι η εξής. Λαμβάνοντας υπόψη μια συλλογή διαφορετικών άρθρων και μια ομάδα κατηγοριών, να τεθεί κάθε άρθρο στην κατάλληλη κατηγορία. Μπορούμε να πάρουμε ένα έγγραφο και για μια δεδομένη κατηγορία να αποφασίσουμε εάν το έγγραφο είναι ή όχι μέλος εκείνης της κατηγορίας. Πρώτα οι λέξεις κλειδιά επιλέγονται για να βοηθήσουν τη διάκριση μεταξύ των κατηγοριών. Οι μεταβλητές στο LP αντιστοιχούν σε αυτές τις λέξεις. Για κάθε άρθρο κάθε λέξη κλειδί μετριέται για να πάρει τη συχνότητά της σε εκείνο το άρθρο. Το διάνυσμα αυτών των συχνοτήτων καθορίζει ένα σημείο στο χώρο των n διαστάσεων. Οι συνιστώσες του διανύσματος αντιστοιχούν στις μεταβλητές ενός γραμμικού προβλήματος. Για τις περισσότερες ομάδες λέξεων το προκύπτον ταμπλώ θα είναι αραιό. Εάν αντί της χρήσης μεμονωμένων λέξεων αθροίσουμε τις ομάδες λέξεων, το πρόβλημα θα γίνει μικρότερο και πυκνότερο. Η ομαδοποίηση είναι γνωστή ως συμπίεση και εξαγωγή χαρακτηριστικών

γνωρισμάτων. Λύνουμε το προκύπτον πυκνό γραμμικό πρόγραμμα για να βρούμε ένα υπερεπίπεδο που χωρίζει τα σημεία της κατηγορίας από τα σημεία εκτός αυτής της κατηγορίας.

Η σχέδιαση ψηφιακού φίλτρου [Steiglitz, 1992] και **εξάλειψη θορύβων στις εικόνες** [Mallat, 1999 pg. 419] δίνουν αφορμή για άλλες πυκνές εφαρμογές οι LP χαλαρώσεις προβλημάτων σχεδίασης μηχανών [Uma, 2000] είναι μια άλλη πυκνή εφαρμογή. Η ιδέα αυτής της εφαρμογής είναι η σχεδίαση διαφόρων στόχους με τέτοιο τρόπο ώστε να ελαχιστοποιηθεί ο συνολικός χρόνος. Οι γραμμές αντιστοιχούν στα χρονικά σημεία. Οι μεταβλητές (στήλες) αντιστοιχούν στους στόχους.

Οι Eckstein et al [1995] αναφέρουν άλλες πυκνές εφαρμογές όπως τα πυκνά κύρια προβλήματα που παράγονται μερικές φορές από την Dantzig-Wolfe ή την Benders διάσπαση, η σχεδίαση ψηφιακού φίλτρου, η ανάλυση δεδομένων και η ταξινόμηση και ο οικονομικός σχεδιασμός ενεργειών.

5.11 Πρότυπα και ανάλυση

Ο παραδοσιακός Simplex αλγόριθμος, είναι αλγόριθμος βελτιστοποίησης με πηδήματα μεταξύ παρακείμενων κορυφών. Το κόστος υπολογισμού του είναι $O(kM(N+M))$, όπου k αντιπροσωπεύει το πλήθος των κορυφών, N αντιπροσωπεύει τον αριθμό μεταβλητών και M αντιπροσωπεύει τον αριθμό των ανισοτικών περιορισμών. Παρακάτω περιγράφουμε την παράλληλη μέθοδο του αλγορίθμου Simplex.

5.11.1 Παράλληλη μέθοδος του αλγορίθμου Simplex (PLPRG)

Input:

$$\text{TAB} = \begin{bmatrix} A & I & B \\ q & 0 & 0 \end{bmatrix}$$

M - Ο αριθμός περιορισμών

N - Ο αριθμός των αγνώστων

A - Ο m επί n πίνακας συντελεστών των περιορισμών

B - Ένα διάνυσμα μήκους m αποτελούμενο από τη δεξιά πλευρά των περιορισμών.

Τα συστατικά του B πρέπει όλα να είναι μη αρνητικά.

q - - Ένα διάνυσμα μήκους n αποτελούμενο από τους συντελεστές της αντικειμενικής συνάρτησης.

I - Μία m επί m μοναδιαία μήτρα.

Output:

q[M+N] - Η αντικειμενική τιμή

iter Ο αριθμός των επαναλήψεων

t - Ο συνολικός χρόνος

Variables:

ITASK - Η τάξη του επεξεργαστή

NPES - Ο αριθμός επεξεργαστών

Begin

1- for $0 \le i < M + N$ do

In processor 0

Set *q[i]:=TAB[M][i]*

for $0 \le k < NPES$ pardo

{ for $0 \le i < M / NPES$ do

For $0 \le j < M + N$ do

Set *C[i][j]:=TAB[k*M/ NPES+i]*

For $0 \le j < M / NPES$ do Set *b[j] := B[j][M+N]*

2- In processor 0

for $0 \le i < N$ do

search i (where $q[i]<0$) Set $column := i$

if failure then Goto (10)

3- for $0 \le k < NPES$ pardo

$X:= min(b[j]/C[j][column],\ 0 \le j < M / NPES)$

Set row_no:=j

4- for $0 \le k < NPES$ pardo

Send (*X,ITTASK*) to processor 0

In processor 0

Sarch *ITASK* which corresponds to *min (X)*

min_L:=ITASK

5- From processor *min_L*

Send row *g:=C[row_no][:]* to all processor

Send variable *h:=b[[row_no]* to all processor

6- for $0 \le k < NPES$ pardo

For $0 \le i < M / NPES$ do{

Set *a:=C[i][column]/g[column]*

For $0 \le j < M + N$ do

Set *C[i][j]:=C[i][j]-a*g[j]*

Set *b[i]:=b[i]-a*h*

}

7- In processor *min_L*

for $0 \le j < M + N$ do{

set *C[row_no][j] := g[j]/g[column]*

set *b[row_no] := h / g[column]*

```
                }
8-      in processor 0
                a:=q[column]/g[column]
                For 0 ≤ i < M + N  do {
                        Set q[i] := q[i]-a*g[i]
                        Set q[N+M] := q[M+N] - a*h
                }
9-      goto (2)
10-     for 0 ≤ i < M + N do
                In processor 0
                        Set TAB[M][i] := q[i]
        for 0 ≤ k < NPES    pardo{
                    for 0 ≤ i < M / NPES  do
                        For 0 ≤ j < M + N do
                                Set  TAB[k*M/NPES+i][j] := C[i][j]
                    for 0 ≤ j < N / NPES  do
                        Set TAB[j][M+N] := b[j]
        }
End.
```

5.11.2 Συγχρονισμένες παράλληλες περιστροφές

Η γενική προσέγγισή μας είναι να διαιρέσουμε τις *m* γραμμές του προβλήματος μεταξύ των NPES επεξεργαστών. Δεν υπάρχει καμία επικάλυψη μεταξύ των γραμμών. Επομένως το b διανύσμα, που κρατά τις τιμές του δεξιού μέλους διαιρείται. Ο κύριος επεξεργαστής διαβάζει το αντικειμενικό διάνυσμα q. Σε κάθε επανάληψη, ο κύριος επεξεργαστής υπολογίζει τον καλύτερο υποψήφιο για τη στήλη περιστροφής μεταξύ των

στηλών του, χρησιμοποιώντας τον κανόνα περιστροφής του Danzig. Ο κύριος επεξεργαστής μεταδίδει (στήλη) το δείκτη της στήλης περιστροφής σε όλους τους επεξεργαστές. Κάθε επεξεργαστής υπολογίζει το διάνυσμα *in*=(*X,ITASK)* και το στέλνει στον κύριο επεξεργαστή, έτσι ώστε *X* είναι η τοπική ελάχιστη αναλογία και ITASK είναι η τάξη του επεξεργαστή. Ο κύριος επεξεργαστής εξετάζει τον κερδίζοντα επεξεργαστή (*min_L*) δηλαδή όποιο έχει το ελάχιστο λόγο. Ο επεξεργαστής που επιλέγεται θέτει *g = row_no* και *h = b*[*row_no*] και στέλνει g και h σε όλους τους επεξεργαστές. Με τη χρησιμοποίηση της γραμμής περιστροφής g κάθε επεξεργαστής κάνει περιστροφή στη δική του γραμμή. Αυτή η περιστροφή είναι το τελευταίο βήμα της επανάληψης Η επανάληψη επαναλαμβάνεται μέχρι το πρόβλημα να γίνει βέλτιστο, αδύνατο ή να ανιχνευθεί απεροριστία.

Το βήμα 6 στον αλγόριθμο (PLPRG) όταν προστίθεται ένα πολλαπλάσιο της γραμμής περιστροφής σε άλλη γραμμή, κάθε επεξεργαστής επεξεργάζεται τις γραμμές του, και έτσι η εργασία σχεδόν ομοιόμορφα κατανέμεται τους επεξεργαστές. Όλος ο χρόνος υπολογιστών, σχεδόν 100% του χρόνου υπολογιστών, ξοδεύεται στο βήμα 6 όταν το μέγεθος προβλήματος είναι μεγάλο.

5.12 Το πειραματικό περιβάλλον και αποτελέσματα

5.12.1 Αρχιτεκτονική

Τα παράλληλα πειράματα εκτελέσθηκαν σε μία 8-κόμβη Pentium III συστάδα διπλοπύρηνων-SMP που διασυνδέονταν με 100Mbps FastEthenet. Κάθε κόμβος έχει δύο Pentium III CPUs των 800 MHz, 256 MB RAM, 16KB L1 I Cache, 16 KB L1 D Cache, 256 KB L2 Cache, και τρέχει Linux με τον πυρήνα 2.4.26. Χρησιμοπήσαμε την MPI υλοποίηση MPICH v.1.2.6, που διαμορφώνεται κατάλληλα για μια συστάδα SMP. Η χρήση αυτής της μηχανής παρέχεται από το Εθνικό Τεχνικό Πανεπιστήμιο της Αθήνας, Σχολή Ηλεκτρολόγων Μηχανικών και μηχανικώνΥπολογιστών.

5.12.2 Υπολογιστικά αποτελέσματα

Ο αλγόριθμος που περιγράφεται στην παράγραφο της 5.11.1 έχει εφαρμοσθεί πειραματικά. Σε αυτήν την ενότητα περιγράφουμε τα αριθμητικά πειράματα και περιγράφουμε τα υπολογιστικά αποτελέσματά μας, τα οποία καταδεικνύουν επιτάχυνση σε σχέση με τον παραδοσιακό αλγόριθμο Simplex στα πυκνά γραμμικά προγράμματα.

Μέγεθος προβλήματος (200×200)

No.processors	niter	Time(secs.)	Secs./niter	Speed up
1	382.1	1.9570822	0.0051219	1
2	382.1	1.0086106	0.0026397	1.9403744
4	382.1	0.9658432	0.0025277	2.0262939

Πίνακας 1:

Μέγεθος προβλήματος (300×300)

No.processors	niter	Time(secs.)	Secs./niter	Speed up
1	636.4	8.9317738	0.0140348	1
2	636.4	4.8551093	0.007629	1.8396648
4	636.4	2.9004258	0.0045576	3.0794698
6	636.4	2.4306677	0.0038194	3.6746173

Πίνακας 2:

Μέγεθος προβλήματος (400×400)

No.processors	niter	Time(secs.)	Secs./niter	Speed up
1	897	22.34042	0.0249057	1
2	897	12.416323	0.0138421	1.7992782
4	897	7.3259092	0.0081671	3.0495081
8	897	4.5529641	0.0050758	4.9067858

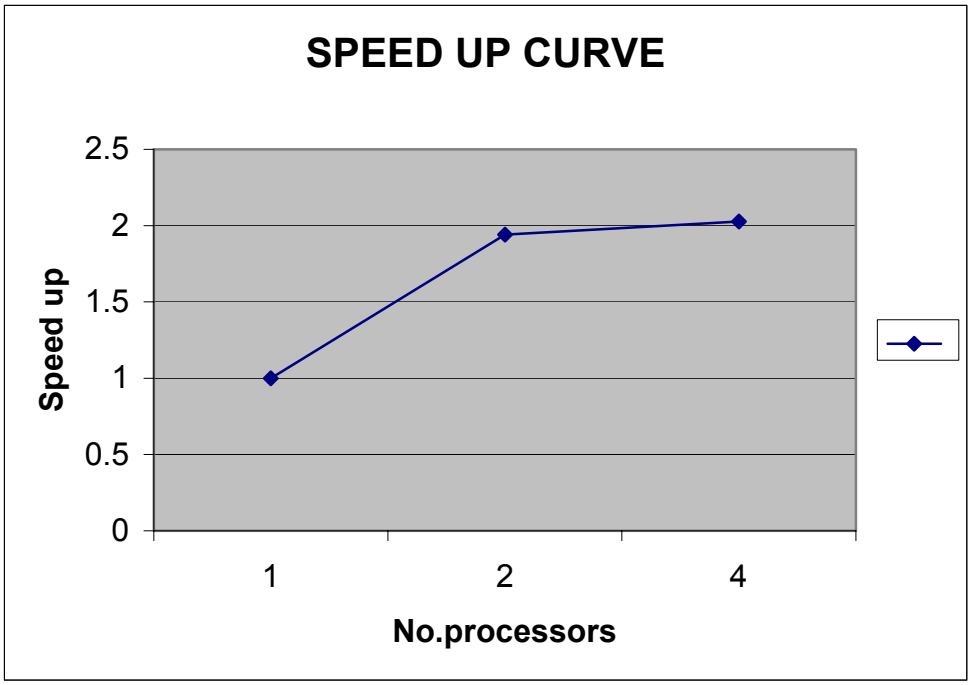
Πίνακας 3:

Τρέξαμε συνολικά 30 τυχαία πυκνά προβλήματα. Τα προβλήματα που εξετάσαμε ήταν μικρά και μεσαία στο μέγεθος. Τα πυκνά γραμμικά προβλήματα βελτιστοποίησης που έχουν λυθεί είναι της μορφής

$$\begin{aligned} \max \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0, \end{aligned}$$

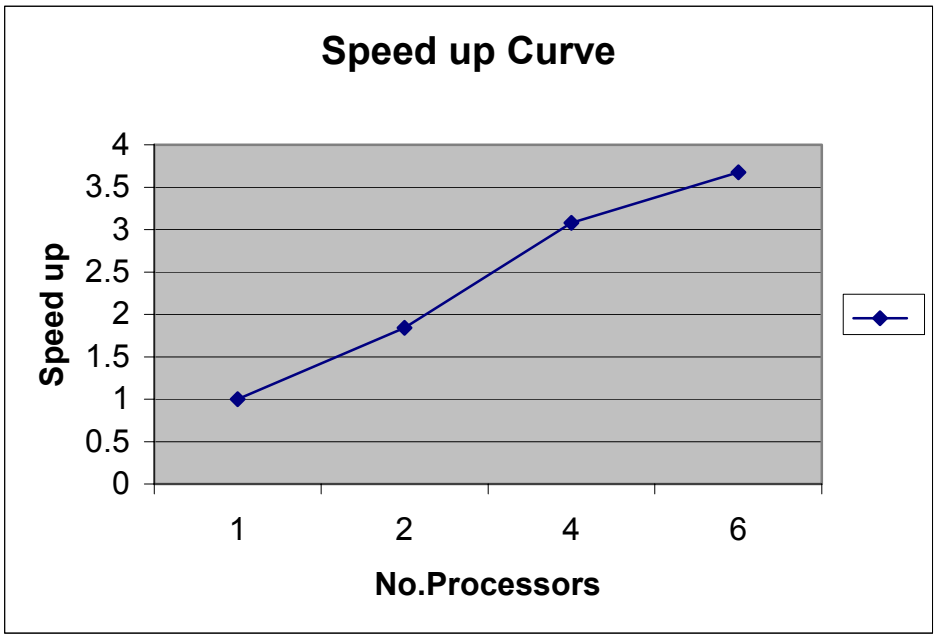
όπου $A \in \mathbb{R}^{m \times n}, c, x \in \mathbb{R}^n, b \in \mathbb{R}^m$ και T δείχνει την αναστροφή. Στην υπολογιστική μελέτη μας χρησιμοποιούμε την περίπτωση $n \times n$, που περιλαμβάνει τρεις διαφορετικές κατηγορίες προβλημάτων που αντιστοιχούν στις τιμές n = 200, 300, 400; κάθε μια από αυτές τις κατηγορίες περιέχει δέκα τυχαία πυκνά γραμμικά προγράμματα. Το εύρος των τιμών που χρησιμοποιήθηκαν για τα τυχαία γραμμικά προγράμματα είναι $c \in$ [1 500] , $A \in$ [-700 1800] , r = 7 και center = 35.

Για κάθε μέγεθος του προβλήματος συγκεντρώνουμε μερικές στατιστικές όσον αφορά τα LPs που χρησιμοποιήθηκαν στην υπολογιστική μας μελέτη και πληροφορίες για την απόδοση του παραδοσιακού Simplex αλγορίθμου. Οι στήλες των πινάκων 1-3 περιέχουν το μέγεθος των προβλημάτων, το πλήθος των επεξεργαστών *no.processor*, το μέσο αριθμό επαναλήψεων *niter* και το μέσο χρόνο CPU *Time(secs.)*.



Σχήμα 5.12.1 Καμπύλη επιτάχυνσης για το μέγεθος προβλήματος (200×200)

Από τα Σχήματα 5.12.1 μπορούμε να δούμε ότι όσο ο αριθμός επεξεργαστών αυξάνεται (μέχρι 4 επεξεργαστές) αυξάνεται και η επιτάχυνση. Τα πειράματά μας δείχνουν ότι εάν αυξήσουμε τον αριθμό επεξεργαστών επάνω από 4 επεξεργαστές δεν αυξάνεται η επιτάχυνση επειδή ο χρόνος επικοινωνίας είναι περισσότερος από τον υπολογιστικό χρόνο για το μέγεθος προβλήματος (200 × 200).



Σχήμα.5.12.2 Καμπύλη επιτάχυνσης για το μέγεθος προβλήματος (300×300)

Από τα Σχήματα 5.12.2 μπορούμε να δούμε ότι όσο ο αριθμός επεξεργαστών αυξάνεται (μέχρι 6 επεξεργαστές) αυξάνεται και η επιτάχυνση. Τα πειράματά μας δείχνουν ότι εάν αυξήσουμε τον αριθμό επεξεργαστών επάνω από 6 επεξεργαστές δεν αυξάνεται η επιτάχυνση επειδή ο χρόνος επικοινωνίας είναι περισσότερος από τον υπολογιστικό χρόνο για το μέγεθος προβλήματος (300 × 300). Μπορούμε επίσης να δούμε ότι όταν αυξάνουμε το μέγεθος προβλήματος από (200 × 200) σε (300 × 300) έχουμε επιτάχυνση μέχρι 6 επεξεργαστές.

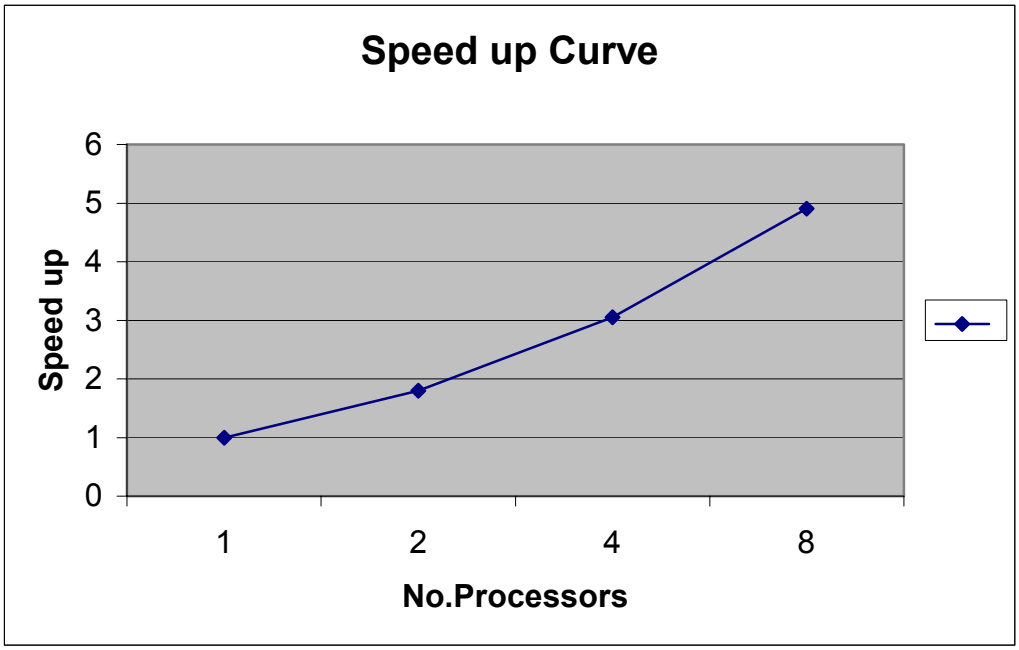
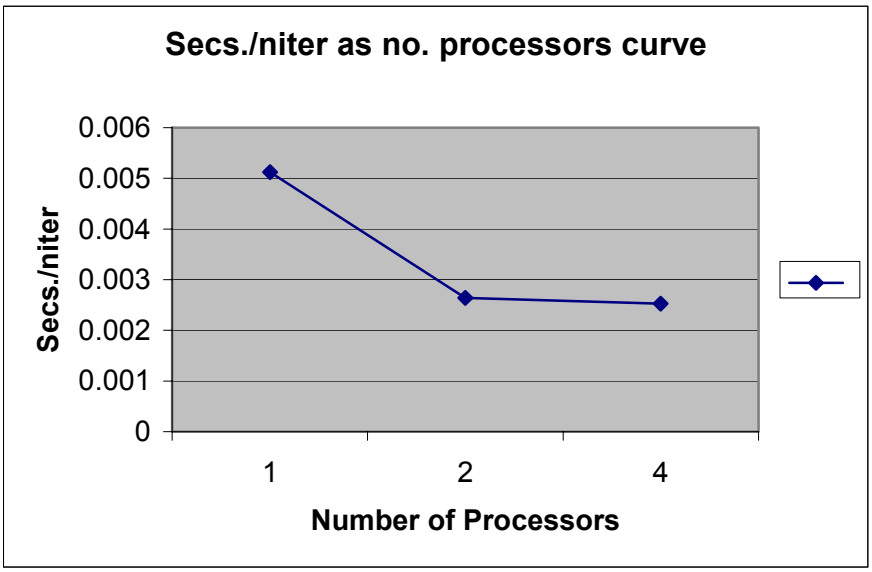
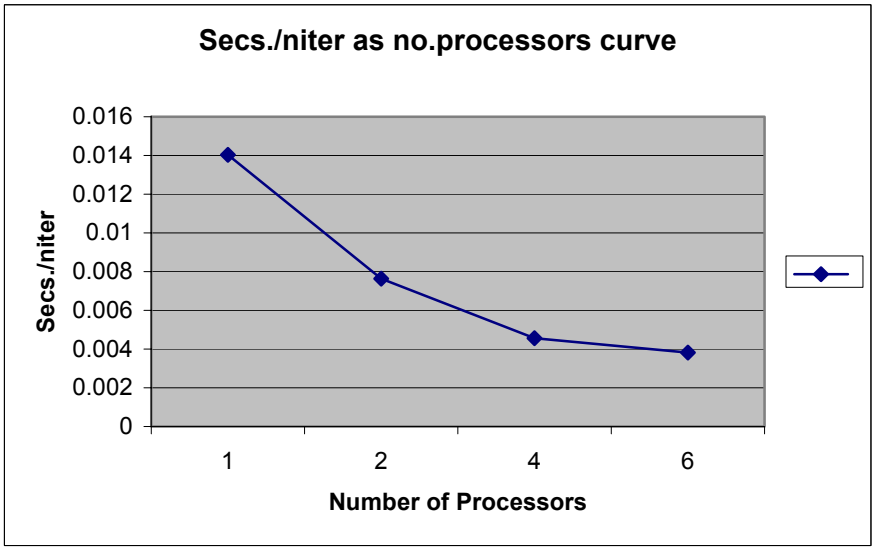


Fig.5.12.3 Καμπύλη επιτάχυνσης για το μέγεθος προβλήματος (400×400)

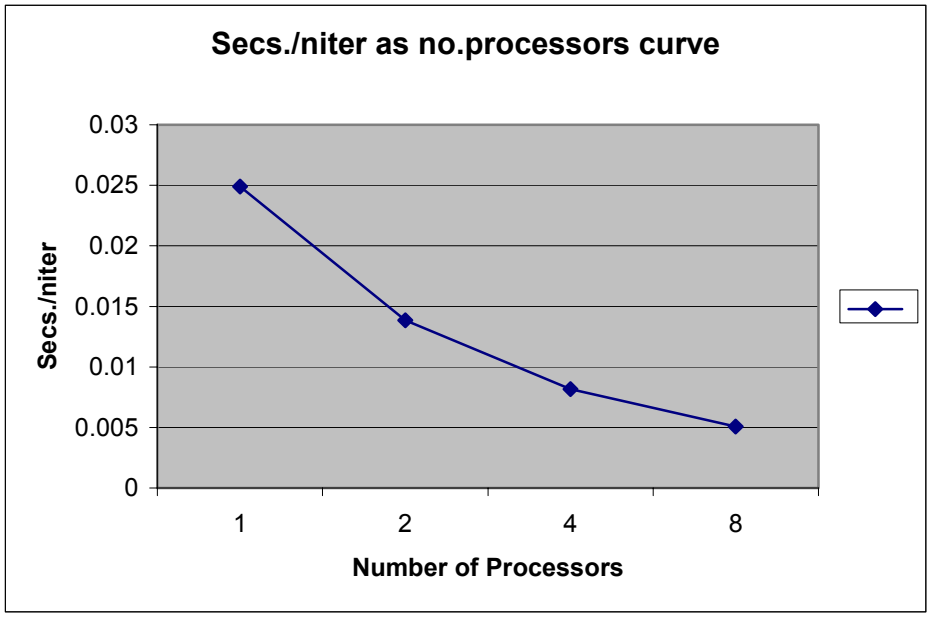
Από τα Σχήματα 5.12.3 μπορούμε να δούμε ότι όσο ο αριθμός επεξεργαστών αυξάνεται (μέχρι 8 επεξεργαστές) αυξάνεται και η επιτάχυνση. Τα πειράματά μας δείχνουν ότι εάν αυξήσουμε τον αριθμό επεξεργαστών επάνω από 8 επεξεργαστές δεν αυξάνεται η επιτάχυνση επειδή ο χρόνος επικοινωνίας είναι περισσότερος από τον υπολογιστικό χρόνο για το μέγεθος προβλήματος (400×400). Μπορούμε επίσης να δούμε ότι όταν αυξάνουμε το μέγεθος προβλήματος από (300×300) (400×400) έχουμε επιτάχυνση μέχρι 8 επεξεργαστές.



Σχήμα 5.12.4 Μέσος χρόνος εκτέλεσης ανά επανάληψη ως αριθμό επεξεργαστών για το μέγεθος προβλήματος (200×200)



Σχήμα 5.12.5 Μέσος χρόνος εκτέλεσης ανά επανάληψη ως αριθμό επεξεργαστών για το μέγεθος προβλήματος (300×300)



Σχήμα 5.12.6 Μέσος χρόνος εκτέλεσης ανά επανάληψη ως αριθμό επεξεργαστών για το μέγεθος προβλήματος (400×400)

Τα Σχήματα 5.12.6, 5.12.5 και Fig. 5.12.4 παρουσιάζουν τη σχέση μεταξύ του μέσου όρου του χρόνου εκτέλεσης ανά επανάληψη και τον αριθμό επεξεργαστών για διαφορετικό μέγεθος προβλημάτων (200 × 200) (300 × 300) και (400 × 400), αντίστοιχα. Από αυτά τα σχήματα μπορούμε να δούμε ότι όσο ο αριθμός επεξεργαστών αυξάνεται, τόσο ο μέσος όρος του χρόνου εκτέλεσης ανά επανάληψης μειώνεται.

ΚΕΦΑΛΑΙΟ 6

ΣΥΖΗΤΗΣΗ ΚΑΙ ΠΡΟΤΑΣΕΙΣ

6.1 Εισαγωγή

Σε αυτό το κεφάλαιο συζητάμε και συνοψίζουμε μερικά προβλήματα και προτάσεις ώστε να επεκτείνουμε και να γενικεύσουμε την εργασία της διατριβής.

6.2 Βελτιώσεις EPSA

Στο κεφάλαιο τέσσερα, παρουσιάσαμε μια υπολογιστικά ελκυστική προσέγγιση στην επίλυση των τυχαίων αραιών προβλημάτων γραμμικού προγραμματισμού. Η υπολογιστική μελέτη δείχνει ότι όσο το μέγεθος του πρόβληματος αυξάνεται και η πυκνότητα του προβλήματος μειώνεται, η συχνότητα που ο EPSA ξεπερνά τον Simplex αλγόριθμο αυξάνεται. Κατά την άποψή μας αυτό είναι μια καλή πρακτική απόδοση που δίνει πολλές υποσχέσεις σε αυτήν την προσέγγιση. Η πειραματική υλοποίηση του αλγορίθμου EPSA σε τυχαία γραμμικά προβλήματα απέδειξε ότι ο αλγόριθμος είναι γρήγορος, υπάρχει όμως ακόμα περιθώριο για περαιτέρω βελτίωσή του. Αυτές οι πιθανές βελτιώσεις θα αποτελέσουν το αντικείμενο της μελλοντικής έρευνάς μας.

Κατά την άποψή μας, οι Simplex τύπου αλγόριθμοι θα συνεχίσουν να εφαρμόζονται στα πρακτικά γραμμικά προβλήματα προγραμματισμού επειδή υπάρχουν πολλές βιομηχανικές εφαρμογές. Επίσης, μπορούμε να βελτιώσουμε την πρακτική απόδοση τον εξωτερικών σημείων τύπου Simplex αλγορίθμο EPSA όταν τρέχει στα μετροπροβλήματα της συλλογής NETLIB.

6.3 Οι βελτιώσεις αντιστροφής της βάσης

Στο κεφάλαιο 3 παρουσίαστηκαν δύο διαφορετικά σχήματα για την ανανέωση της αντίστροφης βάσης. Τα πιο γνωστά σχήματα είναι (i) η μορφή γινομένου του αντιστρόφου (PFI), και (ii) ένα πρόσφατα αναπτυγμένο από τον Benhamadou 2002. Η υπολογιστική μελέτη του κεφαλαίου 3 δείχνει ότι η τροποποίηση της μορφή γινομένου του αντιστρόφου είναι 1,60 φορές γρηγορότερη από τη μορφή γινομένου του αντιστρόφου στα περισσότερα προβλήματα. Θεωρητικά, και οι δύο μέθοδοι απαιτούν τον ίδιο αριθμό πολλαπλασιασμών και προσθέσεων, δηλαδή η πολυπλοκότητα του PFI και MPFI είναι ίδια $\Theta(m^2)$.

Τώρα έχουμε μια ερώτηση που λέει γιατί έχουμε επιτάχυνση μεταξύ PFI και MPFI; Θα απαντήσουμε σε αυτήν την ερώτηση και θα κάνουμε άλλη μια υπολογιστική μελέτη μεταξύ PFI και MPFI στην οποία δεν εκμεταλλευόμαστε την αραιότητα της Eta μήτρας.

6.4 Βελτιώσεις παράλληλου αλγόριθμου Simplex

Στο κεφάλαιο 5, παρουσιάσαμε τον MPI παράλληλο Simplex αλγόριθμο χρησιμοποιώντας του κανόνα του Dantzig. Δεδομένου ότι οι κανόνες περιστροφής παίζουν έναν σημαντικό ρόλο στην επιλογή των μεταβλητών εισόδου και εξόδου μπορούμε να εφαρμόσουμε τον Simplex αλγόριθμο με διαφορετικούς κανόνες περιστροφής για τα παραδείγματα:

Κανόνας Bland’s,

Κανόνας στοίβας,

Κανόνας ουράς,

Η μέθοδος του πιο πρόσφατα θεωρηθέντος στοιχείου,

Κανόνας μέγιστης αύξησης,

Κανόνας μερικής τιμολόγησης

Κανόνας της πιο κατηφορικής ακμής και

Ο κανόνας περιστροφής EPSA.

Υπάρχουν δύο πλατφόρμες για τον παράλληλο προγραμματισμό οι οποία είναι περιβάλλον κοινής μνήμης και το περιβάλλον κατανεμημένης μνήμης. Μπορούμε να εξετάζουμε τα διάφορα σχήματα ισορροπίας φόρτου σε ένα μοιραζόμενο περιβάλλον μνήμης, χρησιμοποιώντας OpenMP και σε ένα κατανεμημένο περιβάλλον μνήμης, χρησιμοποιώντας MPI.

Παράρτημα

Υλοποίηση

Παραθέτουμε τώρα τον κώδικα για ένα πρόγραμμα που γράφεται στη γλώσσα προγραμματισμού C χρησιμοποιώντας τη βιβλιοθήκη MPI. Το πρόγραμμα λύνει ένα πρόβλημα γραμμικού προγραμματισμού με μη αρνητικές μεταβλητές.

```
/*---------------------------------- MPI Parallel Simplex code
--------------------

This program reads the problem in maximization form as
following:

                        max    c^T x

                        s. t. :  Ax ≤ b

                                 x ≥ 0

where  A ∈ R^(m+n), c,x ∈ R^n and b ∈ R^m .

-----------------------------------------------------------------------------
----------------*/
/* include header file  */
#include <mpi.h>
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define float double
#define  MAXR 2000
#define  MAXC 2000
#define BIGNO  1.0E+35
#define  EPSS   2.0E-16


float
A[MAXR][MAXC],B[MAXR],C[MAXR][MAXC],q[MAXC],b[MAXC]
,TAB[MAXR][MAXC],in[2],out[2],g[MAXC];
float X,CMIN,THRESH,RATIO,h,a,time0,time1;
int N,M,local_M,local_N,i,j,ITASK,JTASK,NPES,mynode,
dest,min_L,dest,iter;
int BASIS[MAXR],MROW,LIM,IPHASE,column,row_no,IIP,K;
FILE* fin;
int main(int argc,char *argv[]){
```

```
MPI_Status status;
/* Starts MPI processes ... */
MPI_Init(&argc,&argv);                                  /* starts
MPI                  */
MPI_Comm_rank(MPI_COMM_WORLD, &ITASK);  /* get
current process id     */
MPI_Comm_size(MPI_COMM_WORLD, &NPES);     /* get
number of processes */
time0=0.0;
if(ITASK==0){
       fin=fopen("input0300.txt","r");
    {
       /* Enter M and N */
       fscanf(fin,"%d %d ",&M,&N);
       /* input for coefficient of A   */
        for (i=0;i<M;i++){
          for (j=0;j<N;j++) fscanf(fin,"%le",&A[i][j]);
        }
        /* input for right hand side values   */
```

```
        for (i=0;i<M;i++)  fscanf(fin,"%le",&B[i]);

       /* input for coefficients of objective function  */
        for (j=0;j<N;j++) fscanf(fin,"%le ",&q[j]);
     }
fclose(fin);
time0=MPI_Wtime();
 /* INITIALIZE SIMPLEX TABLEAU        */
                  for(i=0;i<M;i++)
                        for (j=0;j<N+M+1;j++)
                              TAB[i][j]=0.0;

/*LOAD A INTO UPPER LEFT HAND CORNER OF TABLEAU
*/
                for(i=0;i<M;i++)
                   for (j=0;j<N;j++)
                              TAB[i][j]=A[i][j];
```

```
/*LOAD M BY M IDENTITY TO RIGHT OF A AND LOAD B
INTO LAST COLUMN */
                    for(i=0;i<M;i++){
                        TAB[i][N+i]=1.0;
                    }
 /*ROW M+1 CONTAINS -C, INITIALLY */
          for (j=0;j<N;j++)
              q[j]=-q[j];
          for (j=N;j<=N+M;j++)
              q[j]=0.0;
}

MPI_Bcast(&M,1,MPI_INT,0,MPI_COMM_WORLD);
MPI_Bcast(&N,1,MPI_INT,0,MPI_COMM_WORLD);
local_M=M/NPES;
local_N=N/NPES;
MPI_Scatter(TAB,local_M*MAXC,MPI_DOUBLE,C,local_M*MA
XC,MPI_DOUBLE,0,MPI_COMM_WORLD);
```

```
MPI_Scatter(B,local_M,MPI_DOUBLE,b,local_M,MPI_DOUBLE,
0,MPI_COMM_WORLD);
iter=0;
if (ITASK==0){
   THRESH=0.0;
    for (j=0;j<N;j++){
              if (fabs(q[j])>THRESH)
               THRESH=fabs(q[j]);
     }
     THRESH=((1000)*(EPSS)*(THRESH));
}
badr2:
if(ITASK==0){
        column=-1;
        CMIN=-THRESH;
        for (j=0;j<N;j++){
                    if(q[j]<CMIN){
                          CMIN=q[j];
                          column=j;
```

```
            }
        }
}
MPI_Bcast(&column,1,MPI_INT,0,MPI_COMM_WORLD);
if(column==-1) goto badr10;
                    X=BIGNO;
                    row_no=0;
                    for(i=0;i<local_M;i++){
                        if(C[i][column]>THRESH){
                            RATIO=b[i]/ C[i][column];
                            if(RATIO<X){
                              X=RATIO;
                              row_no=i;
                            }
                        }
                    }
in[0]=X;
in[1]=ITASK;
if(ITASK==0){
```

```
        out[0]=X;
        out[1]=0;
}
MPI_Send(&in[0],2,MPI_DOUBLE,0,1,MPI_COMM_WORLD);
        if(ITASK==0){
          for(dest=0;dest<NPES;dest++){

MPI_Recv(&in[0],2,MPI_DOUBLE,dest,1,MPI_COMM_WORLD,
&status);
              if(in[0]<out[0]){
                  out[0]=in[0];
                  out[1]=in[1];
               }
           }
        }
if(ITASK==0){
        min_L=out[1];
}
MPI_Bcast(&min_L,1,MPI_INT,0,MPI_COMM_WORLD);
```

```
if(ITASK==min_L){
  for(j=0;j<N+M;j++)
   g[j]=C[row_no][j];
  g[N+M]=0;
  h=b[row_no];
}
MPI_Bcast(&h,1,MPI_DOUBLE,min_L,MPI_COMM_WORLD);
MPI_Bcast(&g[0],N+M+1,MPI_DOUBLE,min_L,MPI_COMM_W
ORLD);
for(i=0;i<local_M;i++){
   a=C[i][column]/g[column];
   for(j=0;j<N+M;j++)
     C[i][j]=C[i][j]-a*g[j];
   b[i]=b[i]-a*h;
}
if(ITASK==min_L){
   for(j=0;j<N+M;j++)
      C[row_no][j]=g[j]/g[column];
  b[row_no]=h/g[column];
```

```
}
if(ITASK==0){
   a=q[column]/g[column];
   for(i=0;i<N+M;i++){
         q[i]=q[i]-a*g[i];
    }
    q[N+M]=q[N+M]-a*h;
}
iter=iter+1;
time1=MPI_Wtime();
goto badr2;
badr10:
if(ITASK==0){
     printf("\n MASTER  NO.ITERATIO %d\n",iter);
    printf("\n MASTER  objective function %f \n",q[N+M]);
    printf("\n the total time is %f\n",time1-time0);
}
 MPI_Finalize();
  return 0;
```

```
}
```

References

Alexouda, G., Paparrizos, K. (1997). "A comparative computational study with an exterior point simplex algorithm", Proceedings of 4th Balkan Conference on Operational Research, Vol. 1, pp. 12-22.

Andrews, G., and F. Schneider. (1983) "Concepts and notations for concurrent programming", *ComputingSurveys* 15, 3-43.Applied Parallel Research. *Applied Parallel Research*

Anstreicher, K., Terlaky, T. (1994). "A monotonic build-up simplex algorithm for linear programming", *Operations Research*, vol. 42, pp. 556-561.

Badr, E., Paparrizos, K., Samaras, N., Sifaleras, A. (2005) .On the basis inverse of the exterior point simplex algorithm.*17ο Εθνικό Συνέδριο Ελληνικής Εταιρίας ΕπιχειρησιακώνΕρευνών*.

Badr, E., Paparrizos, K., Samaras, N., Baloukas, T. (2006) " Some results on the efficiency of simplex type algorithm" 18ο Εθνικό Συνέδριο Ελληνικής Εταιρίας Επιχειρησιακών Ερευνών.

Badr, E., Paparrizos, K., Samaras, N., Baloukas, T. (2005) " A web-based solver for the single source shortest path problem".18ο Εθνικό Συνέδριο Ελληνικής Εταιρίας Επιχειρησιακών Ερευνών.

Badr, E., Paparrizos, K., Samaras, N. (2006) "On MPI Parallel Simplex Algorithm". 2nd International Computer Engineering Conference.Cairo, EGYPT (Submitted for publication).

Badr, E., Paparrizos, K., Samaras, N. (2006) "On Exterior Point Simplex Algorithm". European Journal of Operational Research (Submitted for publication).

Batoukov, R., and T. Serevik. (1999) “A generic parallel branch and bound environment on a network of workstations”, *Proceedings of High Performance Computing on Hewlett-Packard Systems*, 474-483. (http://www.ii.uib.no/~tors/publications/).

Benaïchouche, M., V.-D. Cung, S. Dowaji, B. Le Cun, T. Mautor, and C. Roucairol. (1996) “Building a parallel branch-and-bound library”, *Lecture Notes in Computer Science* 1054, Springer-Verlag, 201-231.

Bazaraa, S.M., Jarvis, J.J., Sherali, D.H. (1990). “Linear programming and network flows”, 2nd ed., J. Wiley & Sons, Inc., New York, N.Y.

Benhamadou, M. (2002). On the simplex algorithm ‘revised form’, *Advances in Engineering Software*, 33, 769-777.

Bertsimas, D., Tsitsiklis, N.J. (1997). “Introduction to Linear Optimization”, Athena Scientific Press, Belmont, Massachusetts.

Bixby, E.R. (1992). “Implementing the simplex method: The initial basis”, *ORSA Journal on Computing*, Vol. 4, pp. 267-284.

Bixby, E.R. (1994). “Progress in linear programming”, *ORSA Journal on Computing*, Vol. 6(1), pp. 15-22.

Blackfor, L.S., J. Choi, A. Cleary, E. Dázevedo, J. Demmel, I. Dhillon, J. Dongarra, S.Hammarling, G.Henry, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley.(997). *ScaLAPACK User´s Guide*, Society forIndustrial and Applied Mathematics.

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

Borgwardt, H.K. (1982). “Some distribution independent results about the asymptotic

order of the average number of pivot steps in the simplex method", *Mathematics of Operations Research,* Vol. 7(3), pp. 441-462.

Borgwardt, H.K. (1982). "The average number of the pivot steps required by the simplex method is polynomial*", Zeitschrift fur Operational Research, Series A: Theory*, Vol. 26(5), pp. 157-177.

Bosch, Robert and Jason Smith,(1998) "Separating Hyperplanes and the Authorship of the Disputed Federalist Papers," *Mathematical Monthly* August-September.

Bradley, P. S. Usama Fayyad and Olvi Mangasarian,(1999)."Mathematical Programming for Data Mining: Formulations and Challenges," *Informs Journal on Computing* vol. 11 no. 3 pgs. 217-238.

Brüngger, A., A. Marzetta, J. Clausen, and M.Perregaard. ,(1997) "Joining forces in solving large-scale quadratic assignment problems in parallel", *Proceedings of the 11th International Parallel ProcessingSymposium*,418-427. (http://nobi.ethz.ch/group/sppiuk/progress98.html).

Chapman, B., F. Bodin, L. Hill, J. Merlin, G. Viland, and F. Wollenweber. (1999) "FITS – A light-weight integrated programming environment", *Lecture Notes in Computer Science 1685,* 125-134.

Chen, D., Pardalos, P., Saunders, M. (1994). "The simplex algorithm with a new primal and dual pivot rule", *Operations Research Letters*, Vol. 16, pp. 121-127.

Chvatal, V. (1983). " Linear Programming". W. H. Freeman and Company

CPLEX. *(1999)Home page of CPLEX, a division of ILOG,* www.cplex.com

Curet, N.(1974)." A primal-dual simplex method for linear problems". Operations Research Letters;16:233–7

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

Dempster, M.A.H, and R.T. Thompson, (1997) "Parallelization and aggregation of nested Benders decomposition", *Annals of Operations Research* 81, 163-187.(http://www.jims.com.ac.uk/people/mahd.html).

DeRose, L., Y. Zhang, and D. A. Reed, (1998), "SvPablo: A multi-language performance analysis system", *10th International Conference on Computer Performance Evaluation - Modelling techniques and tools*, 352-355.

Dijkstra, E.W. (1968). "Cooperating sequential processes", in *Programming Languages* (F. Genvys, ed.), Academic Press, 43-112.

Dongarra, J., Sullivan, F. (2000). "Guest Editor's Introduction: The Top Ten Algorithms," *Computing in Science and Engineering* January/February pgs.22-23.

Dosios, K., Paparrizos, K. (1997). "Resolution of the problem of degeneracy in a primal and dual simplex algorithm", *Operations Research Letters,* Vol. 20, pp. 45-50.

Eckstein, J., W.E. Hart, and C.A. Philips. (1997). "An adaptable parallel toolbox for branching algorithms", *XVI International Symposium on Mathematical Programming*, Lausanne, p.82.(http://dmawww.epfl.ch/roso.mosiac/ismp97)

Eckstein, Jonathan, I. Boduroglu, L. Polymenakos and D. Goldfarb, (1995) "Data-Parallel Implementations of Dense Simplex Methods on the Connection Machines CM-2," *ORSA Journal on Computing (INFORMS)* vol 7 no. 4 Fall pgs. 402-416.

Elias, D.(1995) "Introduction to parallel programming concepts", Workshop on Parallel Programming on the IBM SP, Cornell Theory Center,. (http://www.tc.cornell.edu/Edu/Workshop)

Etnus Com. *Home page for Etnus Com.*, 1999. (http:www.etnus.com)

Fachat,,A., and K. H. Hoffman, (1997) "Implementation of ensemble-based simulated annealing with dynamic load balancing under MPI", *Computer Physics Communications* 107, 49-53.

Feo, T.A., and M.G.C. Resende. (1995) "Greedy randomized adaptive search procedures", *Journal of GlobalOptimization* 6, 109-133.

Foster, I. (1995). "*Designing and building parallel programs: Concepts and tools for parallel softwareengineering*, Addison-Wesley,.

Flynn, M., (1972) IEEE Trans. Computers 21, 948

Gartner B. (1995) " Randomized optimization by Simplex-type method", Doctoral thesis, Freien University Berlin, Department of Mathematics and Informatics.

Galarowicz, J., and B. Mohr. (1998) "Analyzing message passing programs on the Cray T3Ewith PAT and VAMPIR", *Proceedings of fourth European CRAY-SGI MPP workshop* (H. Lederer and F. Hertwick, eds.)*, IPP-Report des MPI für Plasmaphysik,IPP R/46*, , 29-49. (http://www.kfa=juelich.de/zam/docs/autoren98/galarowicz.html).

Geist, A., A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderman. (1994.). *PVM: Parallel Virtual Machine - A user's guide and tutorial for networked parallel computing*, MIT Press, (www.netlib.org/pvm3/book/pvm-book.html)

Gondzio, J. (1992). "Splitting dense columns of constraint matrix in interior point methods for large-scale linear programming", *Optimization,* Vol. 24, pp. 285-297.

Gondzio, J. (1996). "Multiple centrality corrections in a primal-dual method for linear programming", *Computational Optimization and Applications,* Vol. 6, pp. 137-156.

Gropp, W., E. Lusk, and A. Skjellum. (1994)"*Using MPI: Portable Parallel Programming with the MessagePassing-Interface*, MIT Press,.

Gropp, W., S. Huss-Lederman, A. Lumsdaine, E. Lusk, B. Nitzberg, W. Saphir, and M. Snir. (1998)*MPI: The Complete Reference, Volume 2 - The MPI Extensions*, MIT Press,.

Homer, S., (1997).“Design and performance of parallel and distributed approximation algorithms for maxcut”,*Journal of Parallel and Distributed Computing* 41 , 48-61.

IBM. (1999). “*Users Guide to Parallel OSL”*, SC23-3824,

Karmarkar, K.N. (1984). “A new polynomial time algorithm for linear programming”,

Combinatorica, vol. 4, pp. 373-395.

Klee, V., Minty, G. (1972). “How good is the simplex algorithm?”, In Inequalities

III, Shisha, O. (Ed.), Academic Press, New York, pp. 159-179.

Kliewer, G., and S. Tschöke. (1998) “A general parallel simulated annealing library (parSA) and its applications in industry”, *PAREO’98: First meeting of the PAREO working group on parallel processing in operations research*. (http://www.uni-paderborn.de/~ parsa/)

Kumar, V., A. Grama, A. Gupta, and G. Karypis.(1994).” *Introduction to parallel computing design and analysisof parallel algorithms”*, Benjaming/Cummings,

Lau, L.(1996).”*Implementation of scientific applications in a heterogeneous distributed network”*, PhD. Thesis, University of Queensland, Department of Mathematics,. (http//www.acmc.uq.edu.au/~ll/thesis)

Le Cun, B., C. Roucairol, Benaïchouche, M., V.-D. Cung, S. Dowaji, and T. Mautor. (1995) “BOB: A unified plataform for implementating branch-and-bound like algorithms”, Technical Report RR-95/16, PRiSM Laboratory, Université de Versailles,. (http:/www.prism.uvsq.fr/english/parallel/cr/bob_us.html)

Lemke, E.C. (1954). “The dual method of solving the linear programming problem”.

Naval Research Logistics Quarterly, Vol. 1(1), pp. 36-47.

Linderoth, J., E.K. Lee, and M.W.P. Savelsbergh. (1999) "A parallel linear programming based heuristic for large scale set partitioning problems", Industrial & Systems Engineering Technical Report,. (http://www.isye.gatech.edu/faculty/Eva_K_Lee)

Lustig, J.I., Marsten, J.M., Shanno, D.F. (1994). "Interior point methods for linear programming computational state of the art", *ORSA Journal on Computing*, Vol. 6, pp. 1-14.

Lustig, J.I., Marstenand, E.R., Shanno, F.D. (1992). "On implementing Mehrotra's predictor-corrector interior-point method for linear programming", *SIAM Journal on Optimization,* Vol. 2(3), pp. 435-449.

Mallat, Stéphane, (1999)."*A Wavelet Tour of Signal Processing". 2nd ed.* Academic Press. pg.419.

Maros, I. and Khaliq, M. H. (1999) "Advances in Desine and Implementation of Optimization Software", Technical report, Imperial College London

Martins, S.L., M.G.C. Resende, C.C. Ribeiro, and P. Pardalos. (1999) "A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy", to appear in *Journal of Global Optimization*,.

Martins, S.L., C.C. Ribeiro, and M.C. Souza. (1998) "A parallel GRASP for the Steiner problem in graphs", *Lecture Notes in Computer Science* 1457, 285-297.

Marzetta, A., A. Brüngger, K. Fukuda, and J. Nievergelt. (1999) "The parallel search bench ZRAM and its applications", *Annals of Operations Research 90*, 45-63. (http://www.baltzer.nl/anor/contents/1999/90.html)

Merlin, J., S. Baden, S. Fink, and B. Chapman. (1999) "Multiple data parallelism with HPF and KeLP", *Journal of Future Generation Computer Systems* 15, 393-405.

Miller, B.P., M.D. Callaghan, J.M. Cargille, J. K. Hollingsworth, R.B. Irvin, K.L. Karavanic, K. Kunchithapadam, and T. Newhall. (1995) "The Paradyn parallel performance tools", *IEEE Computer 28*, 37-46.

Morse, H.S. (1994) *Practical parallel computing*, AP Professional.

Mousa, M.,((2005). "Parallel algorithms for the construction of special subgraphs". Ph.D. Thesis, Karlsruhe University, Karlsruhe, Germany.

MPI Forum. (1994) "A Message-Passing Interface Standard", *The International Journal of Supercomputing Applications and High Performance Computing* 8, 138-161.

MPI Forum.(1995) "A Message-Passing Interface Standard", (http://www.mpiforum.org/docs/docs.html)

Murty, K.G. (1983). "Linear Programming". Wiley, New York.

Murty, G.K., Fathi, Y. (1984). "A feasible direction method for linear programming", *Operations Research Letters,* Vol. 3(3), pp. 121-127.

Nieplocha, J., R. J. Harrison, and R. J. Littlefield. "Global arrays: (1994) A portable "shared-memory" programming model for distributed memory computers", *Proceedings of Supercomputing 94*, 340-349. (http://www.tc.cornell.edu/UserDoc/Software/Ptools/global/)

Paparrizos, K. (1988). "A non-dual signature method for assignment problems and a generalization of the dual simplex method for transportation problems", *RAIRO-Operations Research*, Vol. 22(3), pp. 269-289.

Paparrizos, K.(1990).” A generalization of an exterior point simplex algorithm for linear programming problems, *Technical Report,University of Macedonia*

Paparrizos, K. (1991). “An infeasible (Exterior Point) simplex algorithm for assignment problems”, *Mathematical Programming*, Vol. 51, pp. 45-54.

Paparrizos, K. (1993). “An exterior point simplex algorithm for (general) linear programming problems”, *Annals of Operations Research*, Vol. 47, pp. 497-508.
Βιβλιογραφία 217

Paparrizos, K. (1996). “A new primal and dual pivoting rule for the simplex algorithm”, Proceedings of SYMOPIS ’96, pp. 448-453.

Paparrizos, K. (1996). “Exterior point simplex algorithms, simple and short proof of correctness”. Proceedings of SYMOPIS ’96, pp. 13-18.

Paparrizos K.(1997).” Pivoting algorithms generating two paths”. Presentedin ISMP ’97, Lausanne, EPFL Switzerland, Abstracts: 207.

Paparrizos, K. (2001). “MATLAB 6.0”, ZYGOS Publications (In Greek)

Paparrizos, K. Stephanides, G., Samaras, N. (2001). “Improved criteria for identifying optimal basic and nonbasic variables in LP”, *Journal of Computational Analysis and Applications*, Vol. 3(1), pp. 75-82.

Paparrizos, K., Samaras, N., Stephanides, G. (2001). “A new efficient primal dual simplex algorithm”, accepted for publication in *Computers and operations Research.*

Paparrizos, K., Samaras, N., Stephanides, G. (2000). “An efficient simplex type algorithm for sparse and dense linear programs”, accepted for publication in *European Journal of Operational Research*.

Paparrizos, K., Samaras, N., Tsiplidis, K. (2001). "Pivoting algorithms for (LP) generating two paths", *Encyclopedia of Optimization,* Pardalos, P., Floudas, C. (Eds.), Kluwer Academic Publishers, Vol. 4, pp. 302-306.

Paparrizos, K., Samaras, Stephanides G.,(2003)."An efficient simplex type algorithm for sparse and dense linear programs*, European Journal of Operational Research* 148, 323-334.

Paparrizos, K. (1999). "Linear programming, algorithms and applications", ZYGOS Publications, Thessaloniki. (In Greek)

Park, S. (1999). "Linear Programming". Minyong-sa, Seoul

Plastino, A., C.C. Ribeiro, and N. Rodriguez. (1999) "A tool for SPMD application development with support for load balancing", to appear in *Proceedings of the ParCo Parallel Computing Conference*. (http://www.inf.puc-rio.br/~celso).

Protic, J., M. Tomasevic, and V. Milutinovic. (1998) *Distributed Shared Memory*, IEEE Computer Society,.

Provenzano, C. "Pthreads". (http://www.mit.edu:8001/people/proven/pthreads)

Rivera-Gallego, W. (1998) " A genetic algorithm for circulant euclidean distance matrices", *Journal of Applied Mathematics and Computation* 97, 197-208.

Roos, C., Terlaky, T., Vial, J-Ph. (1997). "Theory and algorithms for linear optimization, an interior point approach", J. Wiley & Sons, Inc., New York, NY.

Sewell, G. (2005) " Computation methods of linear algebra" ,Secon Edition, John Wiley & Sons.

Salhi, A., (1998) “Parallel implementation of a genetic-programming based tool for symbolic regression”, *Information Processing Letters* 66, 299-307.

Samaras, N. (2001) “Computational Improvement and Efficient Implementation of Two Paths Pivoting Algorithms”, Ph.D. Thesis, University of Macedonia, Department of Applied Informatics, (in Greek).

Steiglitz, Kenneth, Parks and Kaiser. (1992) “METEOR: A Constraint-Based FIR Filter Design Program.” Institute of Electrical and Electronics Engineers (I*EEE) Transactions on Signal Processing* vol. 40 no. 8

Strang, G.(1990). “Introduction to applied mathematics”. *Wellesley, MA: Wellesley-Cambridge Press*.

Swietanowski, A. (1998)” A New Steepest Edge Approximation for the simplex method for Linear Programming”, Computation Optimization and Application, 10(3), pp 271-281.

Tanenbaum,A., (1992) *Modern Operating Systems*, Prentice-Hall.

Terlaky, T., Zhang, S. (1993). “Pivot rules for linear programming: a survey on recent theoretical developments”. Annals of Operations Research, 46, 203–233.

Thomadakis M. E. and Jyh-Charn Liu J. (1996) “ An Efficient Steepest- Edge Simplex Algorithm for SIMD Computers”, in proc. Of International Conference on Supercomputing, pp 286-293.

Thomadaks, M. E. (1994) “ Implementation and Evalution of Primal and Dual Simplex METHOD with Different Pivot-Selection Techniques in the LPBench Environment”, Texas A&M University, Department of Computer Science.

Uma, R.N., (2000)“Theoretical and Experimental Perspectives on Hard Scheduling Problems,” *PhD Dissertation* Polytechnic University.

Vanderbei, R. J. (2001) “Liner Programming :Foundations and Extennsions”, 2nd ed.,Kluwer Academic Publishers.

Vahalia U. (1996).”*Unix Internals - The New Frontiers”*, Prentice-Hall.

Wolf, K., and O. Kraemer-Furhmann. (1996) “An integrated environment to design parallel object-orientedapplications”, *Lecture Notes in Computer Science* 1123,120-127.

Wright, J.S. (1997). “Primal-Dual Interior Point Methods”, SIAM Press.

Yan, J., and S. Sarukhai. (1996) “Analyzing parallel program performance using normalized performance indices and trace transformation techniques”, *Parallel Computing* 22, 1215-1237.

Zadeh N. (1980) “ What is the worst case behavior of the simplex algorithm?”, Technical Report, Stanford University, Department of Operations Research.

Zakarian, A.A.. (1995).”*Nonlinear Jacobi and □-relaxation methods for parallel network optimization”*. PhD thesis, Mathematical Programming Technical Report 95-17, University of Wisconsin, Madison,. (http://www.cs.wisc.edu/math-prog/tech-reports/)