ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

«Μελέτη αλγορίθμων για περιορισμένη συνέπεια μονοπατιού (restricted path consistency)»

Διπλωματική Εργασία

του

Χατήρα Νικόλαου

Επιβλέπων: Στεργίου Κωνσταντίνος

Αναπληρωτής Καθηγητής Π.Δ.Μ.

Κοζάνη, Οκτώβριος 2015

ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

«Μελέτη αλγορίθμων για περιορισμένη συνέπεια μονοπατιού (restricted path consistency)»

Διπλωματική Εργασία

του

Χατήρα Νικόλαου

Κοζάνη, Νοέμβριος 2015

Ευχαριστίες

Ολοκληρώνοντας πλέον τις προπτυχιακές μου σπουδές, νιώθω την ανάγκη να ευχαριστήσω πολλούς ανθρώπους, όπου χωρίς την στήριξη τους δε θα είχα καταφέρει να φθάσω στο σημείο αυτό.

Το Τμήμα μας και το αντικείμενο σπουδών του ήταν πάντα η πρώτη μου επιλογή, ήταν ένας στόχος για εμένα, τον οποίο κυνήγησα, διεκδίκησα και εν τέλει εκπλήρωσα. Στην διάρκεια της φοιτητικής μου πορείας, διδάχθηκα ένα μεγάλο εύρος γνώσεων πάνω στο αντικείμενο του «Μηχανικού Πληροφορικής & Τηλεπικοινωνιών», γνώσεις οι οποίες αποτελούν εφόδια και παρακαταθήκη για το επαγγελματικό μέλλον που θα ακολουθήσει.

Αναφορικά με τις γνώσεις, λοιπόν, θα ήθελα να εκφράσω ένα μεγάλο ευχαριστώ στους διδάσκοντες καθηγητές του Τμήματος μας, για την άρτια και επιστημονική εκπαίδευση που μου προσέφεραν αυτά τα χρόνια σπουδών. Θεωρώ τον εαυτό μου τυχερό, διότι κατάφερα, πέρα των τυπικών σχέσεων διδάσκοντα-φοιτητή, να αναπτύξω μια άριστη συνεργασία και ανθρώπινη σχέση με κάποιους εξ αυτών, και να τους γνωρίσω, εκτός από επιστήμονες, σε προσωπικό επίπεδο. Ένα Πανεπιστήμιο, μία Σχολή, ένα Τμήμα δεν το καθορίζουν μόνο οι εγκαταστάσεις, το Πρόγραμμα Σπουδών ή οτιδήποτε άλλο, αλλά καταλυτικό ρόλο διαδραματίζει το ανθρώπινο δυναμικό του, τόσο από πλευράς διδασκόντων όσο και από πλευράς φοιτητών. Και στο θέμα του ανθρώπινου δυναμικού, τουλάχιστον, το Τμήμα μας είναι όντως αξιέπαινο.

Όσον αφορά την παρούσα διπλωματική εργασία, θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες στον Κ.Στεργίου, ο οποίος επέβλεπε την εργασία αυτή και μου έδωσε την ευκαιρία να ασχοληθώ με έναν τομέα της Τεχνητής Νοημοσύνης με ευρύ ερευνητικό ενδιαφέρον. Πάντα ήταν εύκαιρος στην επικοινωνία, πολύ συνεργάσιμος και συνεπής στην επίλυση των ερωτήσεων και αποριών που σχηματίζονταν κατά διαστήματα στην πορεία της διπλωματικής μου εργασίας.

Τέλος, δε θα μπορούσα να παραλείψω να ευχαριστήσω την οικογένεια μου, τους γονείς μου και την αδερφή μου, για την ψυχολογική και υλική υποστήριξη που μου παρείχαν όλα αυτά τα χρόνια σπουδών μου και με βοήθησαν, με τον τρόπο τους, να καταφέρω να ολοκληρώσω αυτό το κεφάλαιο της ζωής μου. Τους ευχαριστώ πολύ για τις θυσίες που έκαναν και τους διαβεβαιώνω ότι αυτή η επένδυση τους στο πρόσωπο μου θα αποδώσει τα μέγιστα δυνατά, με πολύ προσωπική προσπάθεια και κόπο.

Περίληψη

Το αντικείμενο της διπλωματικής εργασίας αυτής είναι η μελέτη και η υλοποίηση των αλγορίθμων AC-3(Arc Consistency-3) και RPC(Restricted Path Consistency), για περιορισμένη συνέπεια μονοπατιού. Η περιορισμένη συνέπεια μονοπατιού είναι μια μέθοδος συνέπειας για προβλήματα ικανοποίησης περιορισμών παρόμοια με την συνέπεια τόξου, αλλά όχι τόσο διαδεδομένη. Ο RPC αλγόριθμος που υλοποιείται, αποτελεί προέκταση του αλγορίθμου συνέπειας τόξου AC-3 σε επίπεδο δομών δεδομένων και τρόπου λειτουργίας.

Ο στόχος της παρούσας διπλωματικής εργασίας είναι η υλοποίηση των δύο αλγορίθμων AC-3 και RPC και έπειτα η πειραματική τους σύγκριση, καθώς και η στατιστική τους μελέτη. Η υλοποίηση των αλγορίθμων πραγματοποιήθηκε με τη γλώσσα προγραμματισμού MATLAB. Ως δεδομένα εισόδου χρησιμοποιήθηκαν συγκεκριμένα προβλήματα δυαδικών περιορισμών, κυρίως από τον χώρο των graph-coloring προβλημάτων και όχι μόνο . Η υλοποίηση των αλγορίθμων, σε συνδυασμό με τα αποτελέσματα που προέκυψαν από τα εισαγόμενα προβλήματα , δίνουν τη δυνατότητα να εξάγουμε έγκυρα συμπεράσματα.

Τα αποτελέσματα που μας ενδιαφέρουν αφορούν ως επί το πλείστον την αποτελεσματικότητα και την αποδοτικότητα των αλγορίθμων. Ειδικότερα, με τη χρήση διάφορων μετρικών στον κώδικα των αλγορίθμων, όπως για παράδειγμα το πλήθος διαγραφών τιμών από τα πεδία τιμών των μεταβλητών του προβλήματος ή ο χρόνος εκτέλεσης του αλγορίθμου, ανάλογα με το εισαγόμενο πρόβλημα ικανοποίησης περιορισμών, καταλήγουμε σε χρήσιμα συμπεράσματα για τους αλγορίθμους που πραγματεύεται η παρούσα διπλωματική εργασία.

Στα αρχικά κεφάλαια θα πραγματοποιήσουμε θεωρητική εισαγωγή πάνω στις έννοιες της συνέπειας τόξου, καθώς και της συνέπειας μονοπατιού, με τη χρήση παραδειγμάτων για περαιτέρω κατανόηση, διότι αυτές οι δύο έννοιες αποτελούν το θεμέλιο λίθο των αλγορίθμων AC-3 και RPC που μελετούμε και υλοποιούμε. Στην πορεία θα μελετήσουμε την υλοποίηση των αλγορίθμων στη γλώσσα MATLAB και θα αναλύσουμε τις προγραμματιστικές λεπτομέρειες που χρίζουν σχολιασμό. Τέλος, θα καταλήξουμε σε συμπεράσματα από τα στατιστικά των διάφορων εισαγόμενων προβλημάτων που επιλύουν οι αλγόριθμοι AC-3 και RPC.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

Η παρούσα διπλωματική εργασία πραγματεύεται την μελέτη, την υλοποίηση και την πειραματική σύγκριση των αλγορίθμων AC-3 και RPC για την επίλυση προβλημάτων ικανοποίησης περιορισμών. Σε αυτή την κατηγορία προβλημάτων Τεχνητής Νοημοσύνης ανήκουν αρκετά γνωστά προβλήματα στο χώρο της επιστήμης υπολογιστών, όπως ο χρωματισμός γράφων, το πρόβλημα των *N*-βασιλισσών, λογικά πάζλ, η σχεδίαση και ο προγραμματισμός εργασιών και όχι μόνο. Η λύση σε τέτοιου είδους προβλήματα απαιτεί την ικανοποίηση συγκεκριμένων περιορισμών.

Έχουν προταθεί αρκετές μέθοδοι ελέγχου συνέπειας, οι οποίες σε συνδυασμό με άλλους μηχανισμούς όπως η οπισθοδρομική αναζήτηση “backtrack search”, στοχεύουν στο να μειώσουν, κατά το μέγιστο δυνατό, το χώρο αναζήτησης σε ένα πρόβλημα ικανοποίησης περιορισμών. Αυτό επιτυγχάνεται με τη διαγραφή τιμών από τα πεδία τιμών των μεταβλητών που συμμετέχουν σε περιορισμό μεταξύ τους, όταν αυτές οι τιμές δεν ικανοποιούν τους συγκεκριμένους περιορισμούς. Γνωστές μέθοδοι ελέγχου συνέπειας είναι η συνέπεια κόμβου «node consistency», η συνέπεια ημιτόνου «singleton consistency», η συνέπεια μονοπατιού «path consistency» και η συνέπεια τόξου «arc consistency». Από αυτές, μας απασχολούν στην παρούσα διπλωματική εργασία οι δύο τελευταίες, καθώς ο αλγόριθμος AC-3 που θα ασχοληθούμε επιτυγχάνει συνέπεια τόξου σε προβλήματα δυαδικών περιορισμών και ο αλγόριθμος RPC, ο οποίος αποτελεί τρόπον τινά προέκταση του AC-3, επιτυγχάνει περιορισμένη συνέπεια μονοπατιού σε προβλήματα δυαδικών περιορισμών, έννοιες που θα αναλύσουμε σε παρακάτω κεφάλαια της εργασίας.

Εν κατακλείδι, στην εν λόγω διπλωματική εργασία θα αναπτύξουμε τους δύο παραπάνω αλγορίθμους, χρησιμοποιώντας τη γλώσσα MATLAB, θα κάνουμε ποιοτική και συγκριτική ανάλυση των αποτελεσμάτων τους, θα παραθέσουμε στιγμιότυπα από τον κώδικα και θα συγκρίνουμε την αποδοτικότητα των δύο αλγορίθμων βασιζόμενοι στα στατιστικά μεγέθη που προκύπτουν από πολλαπλές εκτελέσεις των αλγορίθμων με διάφορα εισαγόμενα προβλήματα ικανοποίησης περιορισμών.

1.1 Πλαίσιο Διπλωματικής Εργασίας

Η εν λόγω διπλωματική εργασία περιλαμβάνεται στο γενικότερο πλαίσιο της επίλυσης προβλημάτων ικανοποίησης περιορισμών. Ειδικότερα, μελετούμε και αναπτύσσουμε τον αλγόριθμο AC-3, ο οποίος αποτελεί τον πιο διαδεδομένο αλγόριθμο συνέπειας τόξου "arc consistency" και έπειτα τον αλγόριθμο RPC "restricted path consistency", ο οποίος αποτελεί προγραμματιστική προέκταση του AC-3 και βασίζεται στην συνέπεια μονοπατιού "path consistency".

Δύο είναι τα βασικά σημεία που μπορούμε να συνοψίσουμε την παρούσα διπλωματική εργασία:

Έως και το 5^{ο} κεφάλαιο, γίνεται η θεωρητική εισαγωγή πάνω σε βασικές έννοιες της Τεχνητής Νοημοσύνης γενικότερα, καθώς και ειδικότερες επεξηγήσεις και αναφορές πάνω στο αντικείμενο μελέτης που παρουσιάζουμε. Πιο συγκεκριμένα, ορίζουμε και αναλύουμε τον ορισμό των προβλημάτων με περιορισμούς, με τη χρήση απτών παραδειγμάτων. Επιπλέον, δίνεται λεπτομερής ορισμός της έννοιας «συνέπειας τόξου» στην οποία στηρίζεται ο AC-3. Παρατίθενται σχηματικά παραδείγματα της «συνέπειας τόξου» και επίλυση απλών προβλημάτων βήμα προς βήμα. Στην πορεία, παρουσιάζονται οι αλγόριθμοι AC-3 και RPC, συνοδευόμενοι από ψευδοκώδικα για περαιτέρω κατανόηση τους, καθώς και ανάλυση των τρόπων λειτουργίας τους και των πλεονεκτημάτων και μειονεκτημάτων αυτών.

Στην συνέχεια, στα κεφάλαια 6 και 7 της διπλωματικής εργασίας, παρουσιάζονται η υλοποίηση και εκτέλεση των αλγορίθμων στο περιβάλλον της MATLAB, συνοδευόμενα από τμήματα κώδικα, παρουσίαση στιγμιότυπων εκτέλεσης των αλγορίθμων. Επιπλέον, αναλύονται οι δομές δεδομένων που χρησιμοποιήθηκαν κατά την υλοποίηση των αλγορίθμων και παρατίθενται τα συγκεντρωτικά στατιστικά αποτελέσματα. Στο 7^{ο} και τελευταίο κεφάλαιο ακολουθούν τα τελικά συμπεράσματα που εξήχθησαν από τη μελέτη και την υλοποίηση των δύο αλγορίθμων.

1.2 Οργάνωση Κεφαλαίων

Η δομή της παρούσας διπλωματικής εργασίας ανά κεφάλαιο, έχει ως εξής:

- Στο Κεφάλαιο 1 παρατίθεται η εισαγωγή της διπλωματικής εργασίας
- Στο Κεφάλαιο 2 παρουσιάζονται και αναλύονται βασικά χαρακτηριστικά των προβλημάτων ικανοποίησης περιορισμών, δίνονται οι απαραίτητοι ορισμοί και επιλύονται απλά παραδείγματα
- Στο Κεφάλαιο 3 αναλύονται η έννοια της «συνέπειας τόξου» προκειμένου να μελετηθεί ο αλγόριθμος AC-3, καθώς και οι έννοιες της «συνέπειας μονοπατιού» και της «περιορισμένης συνέπειας μονοπατιού» στις οποίες βασίζεται ο RPC
- Στο Κεφάλαιο 4 μελετάται και αναλύεται ο αλγόριθμος AC-3
- Στο Κεφάλαιο 5 μελετάται και αναλύεται ο αλγόριθμος RPC
- Στο Κεφάλαιο 6 αναλύεται και παρουσιάζεται εκτενώς ο κώδικας και η εκτέλεση των αλγορίθμων και παρουσιάζονται τα αποτελέσματα των αλγορίθμων
- Στο Κεφάλαιο 7 παρατίθενται τα συμπεράσματα που προέκυψαν από την μελέτη και υλοποίηση των αλγορίθμων

Κεφάλαιο 2

Εισαγωγή στα Προβλημάτων Ικανοποίησης Περιορισμών (CSP)

Το κεφάλαιο αυτό μελετά αρχικά σε θεωρητικό επίπεδο τα προβλήματα ικανοποίησης περιορισμών και στην συνέχεια προχωρά σε πρακτική μελέτη αυτών. Δίνεται ο ορισμός των προβλημάτων ικανοποίησης περιορισμών και παρουσιάζονται τρόποι επίλυσης, προχωρώντας και σε επίλυση ενός παραδείγματος, ώστε να γίνει πλήρης κατανοητή η φύση ενός τέτοιου προβλήματος. Στο τέλος του κεφαλαίου, σημειώνονται προβλήματα από το χώρο της επιστήμης υπολογιστών και όχι μόνο, τα οποία μπορούν να εκφρασθούν και να μοντελοποιηθούν ως προβλήματα ικανοποίησης περιορισμών.

2.1 Ορισμός και Αναπαράσταση των Προβλημάτων Ικανοποίησης Περιορισμών

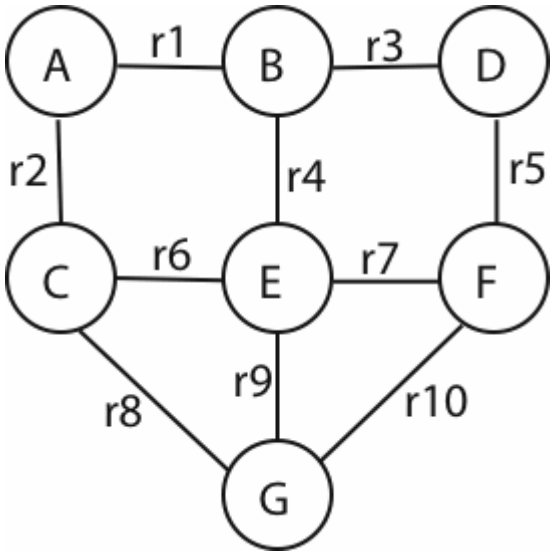
Ένα πρόβλημα ικανοποίησης περιορισμών (CSP) ορίζεται ως μια τριπλέτα $\{X, D, C\}$, όπου $X = \{x_1, x_2, \ldots, x_n\}$ είναι ένα σύνολο n μεταβλητών (variables), $D = \{D(x_1), D(x_2), \ldots D(x_n)\}$ είναι ένα σύνολο πεδίων τιμών (domains), ένα για κάθε μεταβλητή και $C = \{c_1, c_2, \ldots, c_e\}$ είναι ένα σύνολο e περιορισμών. Κάθε περιορισμός C_{ij} αφορά τις μεταβλητές x_i και x_j και προσδιορίζει τους επιτρεπτούς συνδυασμούς τιμών για το υποσύνολο αυτό.

Τα προβλήματα ικανοποίησης περιορισμών μπορούν να κατηγοριοποιηθούν στις εξής κατηγορίες:

- Προβλήματα ικανοποίησης περιορισμών αποτελούμενα από περιορισμούς με μία μόνο μεταβλητή (unary problems), όπως για παράδειγμα $Z > 3$.
- Προβλήματα ικανοποίησης περιορισμών αποτελούμενα από περιορισμούς με δύο μεταβλητές (binary problems), όπως για παράδειγμα το $X - Y > 9$.

- Προβλήματα ικανοποίησης περιορισμών αποτελούμενα από περιορισμούς με n −μεταβλητές, (n-ary problems), όπως για παράδειγμα $Z + 3 \leq Y - X$

Μπορούμε να αναπαραστήσουμε σχηματικά ένα πρόβλημα ικανοποίησης περιορισμών ως ένα γράφημα (Εικόνα 2.1), του οποίου οι κόμβοι αποτελούν τις μεταβλητές του προβλήματος και οι ακμές του γραφήματος που ενώνουν τους κόμβους αποτελούν τους δυαδικούς περιορισμούς μεταξύ των μεταβλητών του προβλήματος.



Εικόνα 2.1 *~ Γράφος ενός CSP όπου οι κόμβοι A έως G είναι οι μεταβλητές του προβλήματος και οι ακμές r1 έως r10 αναπαριστούν τους περιορισμούς μεταξύ των μεταβλητών*

2.2 Ταξινόμηση, Στόχος και Τρόποι Επίλυσης ενός CSP

Τα προβλήματα ικανοποίησης περιορισμών μπορούν να ταξινομηθούν με βάση τα ακόλουθα χαρακτηριστικά:

- Διακριτών ή συνεχόμενων μεταβλητών
- Πεπερασμένα ή μη πεπερασμένα πεδία τιμών
- Γραμμικοί ή μη γραμμικοί περιορισμοί
- Μοναδιαίοι, δυαδικοί,…, n-αδικοί περιορισμοί

Στην παρούσα εργασία μας απασχολούν μόνο προβλήματα ικανοποίησης περιορισμών με διακριτές μεταβλητές, πεπερασμένων πεδίων τιμών, γραμμικών και δυαδικών περιορισμών.

Το ζητούμενο στα προβλήματα ικανοποίησης περιορισμών είναι να αναθέσουμε τιμές σε ένα σύνολο μεταβλητών έτσι ώστε να ικανοποιούνται όλοι οι περιορισμοί του προβλήματος. Οι περιορισμοί συμπεριλαμβάνουν συνδυασμούς μεταβλητών οι οποίες λαμβάνουν τιμές από επιτρεπτά πεδία ορισμού.

Ο στόχος στα προβλήματα ικανοποίησης περιορισμών συνήθως είναι να βρούμε μία και μόνο λύση στο πρόβλημα, δεν είναι όμως και ο αποκλειστικός στόχος. Σε άλλες περιπτώσεις, έχουμε ως στόχο την εύρεση μιας προσεγγιστικής λύσης στο πρόβλημα, ή επίσης την εύρεση λύσης η οποία θα ελαχιστοποιεί ή μεγιστοποιεί ένα ποσοτικό μέγεθος του προβλήματος, όπως για παράδειγμα στα προβλήματα βελτιστοποίησης.

Ο τρόπος επίλυσης ενός CSP έγκειται στην ανάθεση τιμών σε κάθε μεταβλητή με τέτοιο τρόπο ώστε να μην παραβιάζεται κανένας περιορισμός μεταξύ των μεταβλητών του προβλήματος. Ένα CSP ονομάζεται συνεπές (consistent) αν έχει μια τουλάχιστον λύση, διαφορετικά ονομάζεται ασυνεπές (inconsistent). Υπάρχει ένα αρκετά μεγάλο εύρος αλγορίθμων που λύνουν ολοκληρωτικά ή μερικά τέτοιας φύσεως προβλήματα, εφαρμόζοντας ο καθένας διαφορετικές τεχνικές, ωστόσο διαφοροποιούνται τόσο σε αποδοτικότητα όσο και σε χωρική και χρονική πολυπλοκότητα. Ενδεικτικά, αναφέρουμε και επεξηγούμε συνοπτικά τους παρακάτω:

1. **Απλή ή χρονολογική υπαναχώρηση (simple or chronological backtracking - BT)**

   Η βασική ιδέα σε κάθε αλγόριθμο υπαναχώρησης είναι να ξεκινήσουμε με μια μερική λύση και να την επεκτείνουμε μέχρι να καταλήξουμε σε μια πλήρη λύση. Ο αλγόριθμος BT ακολουθεί αυτή τη γενική μέθοδο. Επιπρόσθετα, όταν φθάνει σε αδιέξοδο, πάντα υπαναχωρεί στην τελευταία ληφθείσα απόφαση, εξ' ου και το όνομα του. Χαρακτηρίζεται ως πλήρης αλγόριθμος, με χρονική πολυπλοκότητα $O(d^n e)$ όπου d είναι ο μέγιστος αριθμός στοιχείων ενός πεδίου, n είναι ο αριθμός μεταβλητών και e ο αριθμός των περιορισμών, ενώ η χωρική του πολυπλοκότητα είναι $O(nd)$.

2. **Πρώιμος Έλεγχος (Forward Checking - FC)**

Ο αλγόριθμος του πρώιμου ελέγχου στηρίζεται στην ικανοποίηση της εξής συνθήκης: Για κάθε μεταβλητή στην οποία δεν έχει ανατεθεί τιμή, υπάρχει τουλάχιστον μια τιμή στο πεδίο της που είναι συνεπής με τις τιμές που έχουν ανατεθεί σε άλλες μεταβλητές. Ο FC λειτουργεί με τέτοιο τρόπο, ώστε κάθε φορά που μια τιμή v ανατίθεται σε μια μεταβλητή X, εξετάζει κάθε μεταβλητή Y στην οποία δεν έχουν ανατεθεί τιμές και διαγράφει όλες τις τιμές της Y που είναι ασυνεπείς με τη v από το πεδίο της. Αν αυτή η ενέργεια έχει ως αποτέλεσμα το πεδίο της Y να γίνει κενό, τότε η v απορρίπτεται.

3. **Υπαναχώρηση με Άλμα (Backjumping - BJ)**

Ο αλγόριθμος αυτός είναι ένας αλγόριθμος ευφυούς υπαναχώρησης. Όταν φθάνει σε αδιέξοδο (dead-end) για μια μεταβλητή X, ο BJ δεν υπαναχωρεί στην προηγούμενη μεταβλητή όπως ο BT, αλλά στη μεταβλητή που βρίσκεται βαθύτερα στο δένδρο αναζήτησης, την πιο πρόσφατη μεταβλητή δηλαδή, που είχε ως συνέπεια να απαλειφθεί μια τιμή από το πεδίο τιμών της X. Ο BJ υπαναχωρεί με άλμα μόνο όταν βρεθεί σε αδιέξοδο που προκύπτει από τις προηγούμενες αναθέσεις.

4. **Συνέπεια τόξου (Arc Consistency – AC)**

Η συνέπεια τόξου είναι από τους βασικότερους τρόπους διάδοσης περιορισμών. Δεν είναι σε θέση να εξασφαλίσει από μόνη της λύση σε ένα CSP, διότι πετυχαίνει τοπική συνέπεια (local consistency). Χρησιμοποιείται συνδυαστικά με άλλους αλγορίθμους, όπως ο DFS (Depth-First Search) για παράδειγμα, ως προπαρασκευαστικό μέτρο με στόχο να αφαιρεθούν όσο το δυνατόν πιο πολλές τιμές από τα πεδία ορισμών των μεταβλητών, με την προϋπόθεση ότι δεν ικανοποιούν τους περιορισμούς του προβλήματος. Αυτό γίνεται, για να μειωθεί ο χώρος αναζήτησης και να επιτευχθεί ταχύτερα η επίλυση του προβλήματος.

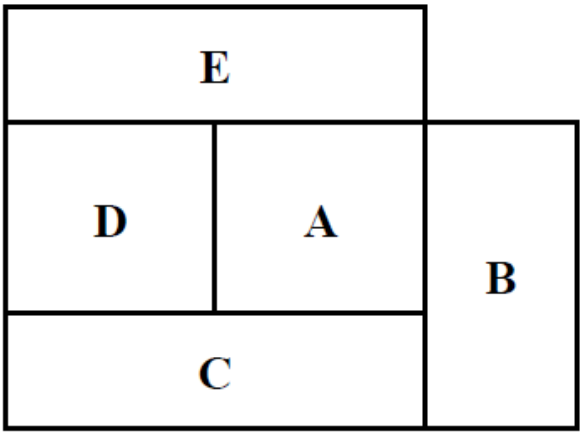
5. **Μηχανισμός των Ελάχιστων Συγκρούσεων (min-conflicts)**

Ο min-conflicts είναι ένας αλγόριθμος τοπικής αναζήτησης ιδιαίτερα διαδεδομένος. Η λειτουργία του βασίζεται στην επιλογή μιας νέας τιμής για μια

μεταβλητή, η οποία τιμή θα προκαλέσει τον ελάχιστο αριθμό συγκρούσεων με την τρέχουσα ανάθεση στις άλλες μεταβλητές. Πιο απλά, η τιμή εκείνη που θα δώσει τον ελάχιστο δυνατό αριθμό μεταβλητών που θα χρειασθεί να «επιδιορθωθούν». Ο αλγόριθμος αυτός είναι πολύ ισχυρός για πολλά προβλήματα ικανοποίησης περιορισμών.

2.3 Παράδειγμα ενός Προβλήματος Ικανοποίησης Περιορισμών

Θα δούμε ένα απλό παράδειγμα map coloring(Εικόνα 2.3.1) για να αντιληφθούμε καλύτερα την έννοια ενός CSP και τον τρόπο επίλυσης τους. Έστω ότι έχουμε τον ακόλουθο χάρτη χωρισμένο σε τμήματα:



Εικόνα 2.3.1 *~ Παράδειγμα CSP για πρόβλημα map coloring πριν την επίλυση του*

Ο στόχος είναι να χρωματίσουμε τον χάρτη χρησιμοποιώντας τα τρία βασικά χρώματα (κόκκινο, μπλε και πράσινο), έτσι ώστε δύο διπλανά τμήματα του χάρτη να μην έχουν το ίδιο χρώμα.

Συνεπώς, οι μεταβλητές του προβλήματος μαζί με τα πεδία τιμών τους έχουν ως εξής:

- A με πεδίο τιμών *Domain(A)* = {Red,Green,Blue}
- B με πεδίο τιμών *Domain(B)* = {Red,Green,Blue}

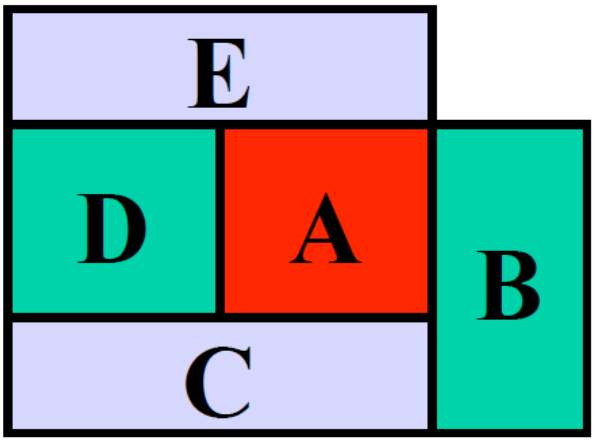
- C με πεδίο τιμών *Domain(C)* = {Red,Green,Blue}
- D με πεδίο τιμών *Domain(D)* = {Red,Green,Blue}
- E με πεδίο τιμών *Domain(E)* = {Red,Green,Blue}

Οι περιορισμοί του προβλήματος είναι οι ακόλουθοι:

$A \neq B$, $A \neq C$, $A \neq E$, $A \neq D$, $B \neq C$, $C \neq D$, $D \neq E$

Το ζητούμενο είναι να βρεθεί μια λύση που θα ικανοποιεί τις παραπάνω συνθήκες και δε θα παραβιάζει κανέναν από τους περιορισμούς του προβλήματος.

Ενδεικτικά, μπορούμε να δώσουμε λύση στο CSP θέτοντας την περιοχή A=red, B=green, C=blue, D=green και E=blue. Επομένως, το map(Εικόνα 2.3.2) θα είναι ως εξής πλέον:



Εικόνα 2.3.2 *~ Επίλυση του προβλήματος map coloring, κάθε περιοχή είναι χρωματισμένη με διαφορετικό χρώμα και καμία από τις γειτονικές περιοχές δε διαθέτει το ίδιο χρώμα*

2.4 Μοντελοποίηση Προβλημάτων CSP σε άλλους τομείς

Τα CSP αποτελούν μοντελοποίηση των προβλημάτων η οποία έχει εφαρμογές και σε άλλους τομείς της σύγχρονης επιστήμης και μπορεί να χρησιμοποιηθεί για να μοντελοποιήσει και πολλά πρακτικά προβλήματα, όπως τα παρακάτω:

- Χρονοπρογραμματισμός Εργασιών (Scheduling)
- Κατάστρωση Προγράμματος (time tabling)
- Ικανοποιησιμότητα στην προτασιακή λογική
- Χρονική Συλλογιστική (Temporal Reasoning)
- Χωρική Συλλογιστική (Spatial Reasoning)
- Ακέραιος, γραμμικός και μη γραμμικός προγραμματισμός
- Δρομολόγηση Οχημάτων (Vehicle Routing)
- Κατανομή Πόρων (Resource allocation)
- Σύνθεση Ανθρώπινης Ομάδας (Manpower rostering)
- Σχεδιασμός Ενεργειών (Planning)
- Κατανομή Συχνοτήτων (Frequency Assignment)

Κεφάλαιο 3

Συνέπεια Τόξου (Arc Consistency) και Συνέπεια Μονοπατιού (Path Consistency)

Στο κεφάλαιο αυτό θα μελετήσουμε αναλυτικά την έννοια της συνέπειας τόξου και την έννοια της συνέπειας μονοπατιού, θα δοθούν οι επεξηγηματικοί ορισμοί τους και θα προβούμε σε περαιτέρω ανάλυση. Επιπρόσθετα, θα παρουσιασθούν και από ένα απλό παράδειγμα για την καθεμία έννοια ξεχωριστά, ώστε να γίνει κατανοητό στα επόμενα κεφάλαια πως αυτές θα χρησιμοποιηθούν στους αλγορίθμους που εξετάζουμε στην παρούσα διπλωματική εργασία.

3.1 Ορισμός Συνέπειας Τόξου

Ας υποθέσουμε ότι K είναι ένα δίκτυο από δυαδικούς περιορισμούς, V είναι το σύνολο των μεταβλητών του δικτύου, D είναι το σύνολο των πεδίων τιμών των μεταβλητών και C το σύνολο των περιορισμών του δικτύου.

Μια τιμή $a_i \in D(x_i)$ παρουσιάζει συνέπεια τόξου εάν για κάθε περιορισμό c_{ij} που συμμετέχει, υπάρχει μια τιμή $a_j \in D(x_j)$ τέτοια ώστε το ζεύγος των τιμών (a_i, a_j) ικανοποιεί τον περιορισμό c_{ij} . Σε αυτήν την περίπτωση, η τιμή a_j καλείται support της a_i. Μια τιμή παρουσιάζει συνέπεια τόξου αν όλες οι μεταβλητές της παρουσιάζουν συνέπεια τόξου. Γενικεύοντας την έννοια της συνέπεια τόξου, ένα πρόβλημα παρουσιάζει συνέπεια τόξου, αν δεν υπάρχει κανένα κενό πεδίο στο σύνολο D και όλες του οι μεταβλητές παρουσιάζουν συνέπεια τόξου.

3.2 Ανάλυση, Εφαρμογή και Πολυπλοκότητα της Συνέπειας Τόξου

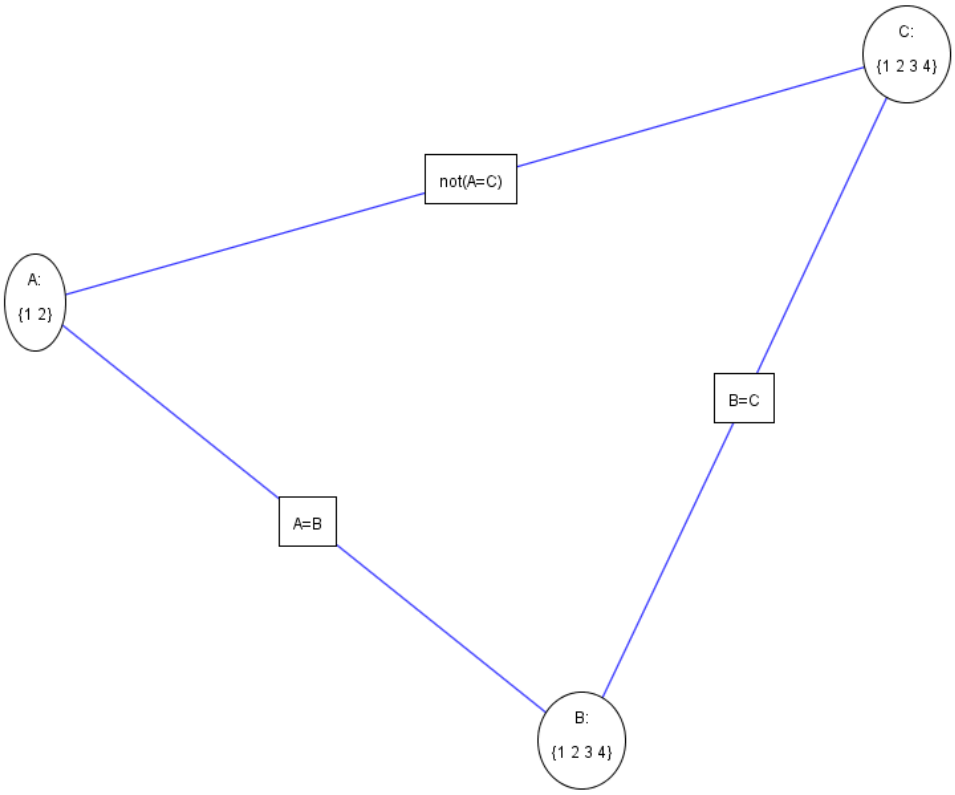
Η ιδέα της συνέπειας τόξου (arc consistency) μας παρέχει μια γρήγορη μέθοδο για τη διάδοση περιορισμών που είναι σημαντικά ισχυρότερη από τον πρώιμο έλεγχο(forward checking) που αναλύσαμε σε προηγούμενο κεφάλαιο. Με την έννοια διάδοση περιορισμού(constraint propagation) αναφερόμαστε στη διάδοση των επιπτώσεων ενός περιορισμού που ισχύει για μια μεταβλητή σε άλλες μεταβλητές. Αν και ο πρώιμος έλεγχος ανιχνεύει πολλές ασυνέπειες, δεν τις ανιχνεύει όλες ωστόσο. Για αυτό εισήχθη η έννοια της συνέπειας τόξου. Με τον όρο «τόξο» εννοούμε ένα προσανατολισμένο τόξο στο γράφημα περιορισμών μεταξύ των κόμβων (μεταβλητών) του γραφήματος.

Ο έλεγχος της συνέπειας τόξου μπορεί να εφαρμοσθεί είτε ως στάδιο προεπεξεργασίας πριν αρχίσει η διαδικασία αναζήτησης είτε ως στάδιο αναζήτησης μετά από κάθε ανάθεση τιμής κατά την αναζήτηση. Και στις δύο περιπτώσεις, η διαδικασία πρέπει να εφαρμόζεται επανειλημμένα μέχρι να μην απομένουν άλλες ασυνέπειες. Αυτό συμβαίνει επειδή, όποτε διαγράφεται μια τιμή από το πεδίο κάποιας μεταβλητής για να αρθεί μια ασυνέπεια τόξου, μια νέα ασυνέπεια τόξου θα μπορούσε να προκύψει στα τόξα που δείχνουν στην μεταβλητή αυτή.

Η πολυπλοκότητα του ελέγχου της συνέπειας τόξου μπορεί να αναλυθεί ως εξής: Ένα πρόβλημα δυαδικών περιορισμών (binary CSP) έχει το πολύ $O(n^2)$ τόξα. Κάθε τόξο (x_k, x_i) μπορεί να εισαχθεί προς επανέλεγχο μόνο d φορές, διότι η X_i έχει το πολύ d τιμές που μπορούν να διαγραφούν, επομένως ο έλεγχος της συνέπειας ενός τόξου μπορεί να γίνει σε χρόνο $O(d^2)$. Άρα, ο ολικός χρόνος της χειρότερης περίπτωσης είναι $O(n^2 d^3)$. Αν και η μέθοδος αυτή είναι σημαντικά πιο δαπανηρή από τον πρώιμο έλεγχο, συνήθως είναι καλύτερη παρά το αυξημένο κόστος.

3.3 Παράδειγμα της Συνέπειας Τόξου

Έχουμε το παρακάτω πρόβλημα ικανοποίησης περιορισμών με τη μορφή γραφήματος:



Οι μεταβλητές είναι: A,B,C

Τα πεδία τιμών είναι: $D(A) = \{1,2\},\ D(B) = \{1,2,3,4\}$ και $D(C) = \{1,2,3,4\}$

Οι περιορισμοί είναι A=B, B=C και A≠C

Θα εφαρμόσουμε συνέπεια τόξου.

Αρχικά, ελέγχουμε τον περιορισμό A≠C

Έχουμε ότι: $D(A) = \{1,2\}$

$D(C) = \{1,2,3,4\}$

$D(B)$ καμία αλλαγή

Έπειτα, ελέγχουμε τον περιορισμό B=C

Έχουμε ότι: $D(B) = \{1,2,3,4\}$

$D(C) = \{1,2,3,4\}$

$D(A)$ καμία αλλαγή

Αμέσως μετά, ελέγχουμε τον περιορισμό A=B

Έχουμε ότι: $D(A) = \{1,2\}$

$D(B) = \{1,2\}$ Διαγραφή των τιμών 3 και 4 από το $D(B)$

$D(C)$ καμία αλλαγή

Επόμενο βήμα, ελέγχουμε τον περιορισμό B=C εκ νέου λόγω της διαγραφής τιμών από το $D(B)$ στο προηγούμενο βήμα

Έχουμε ότι: $D(B) = \{1,2\}$

$D(C) = \{1,2\}$ Διαγραφή των τιμών 3 και 4 από το $D(C)$

$D(A)$ καμία αλλαγή

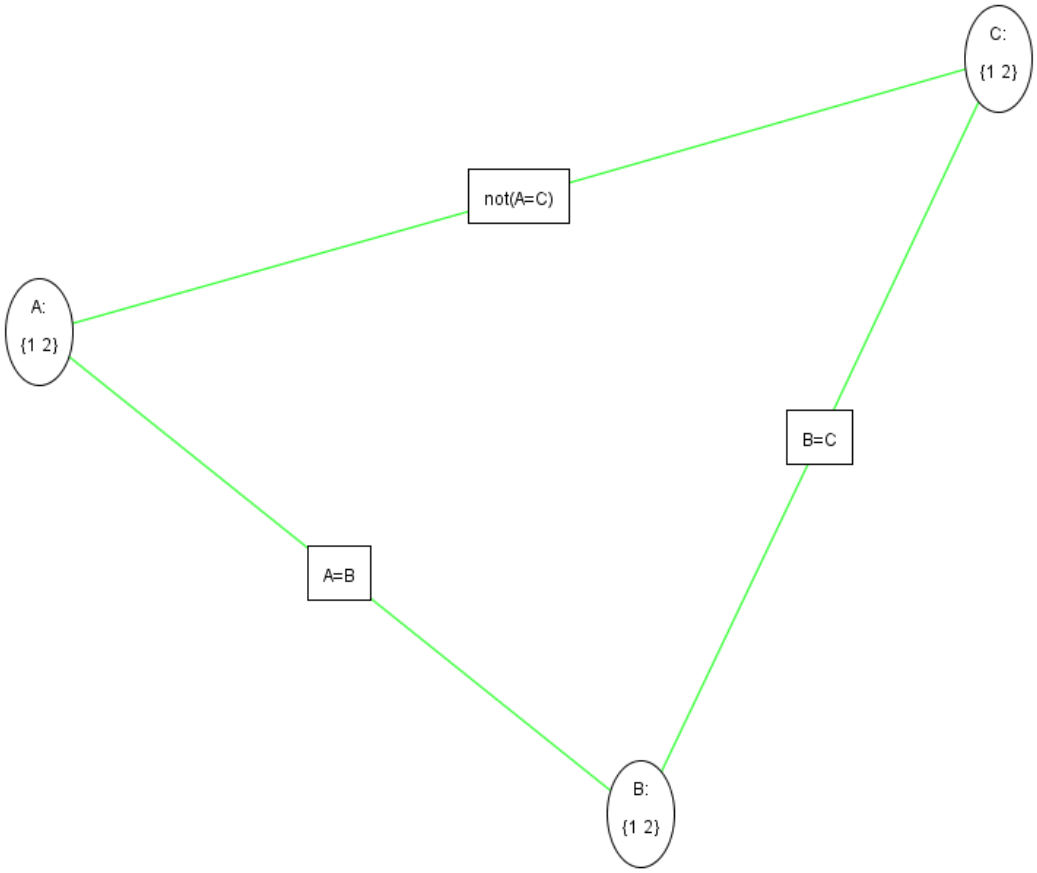
Τελευταίο βήμα, ελέγχουμε εκ νέου τον περιορισμό A≠C εξαιτίας της διαγραφής τιμών από το $D(C)$ στο προηγούμενο βήμα

Έχουμε ότι: $D(A) = \{1,2\}$

$D(C) = \{1,2\}$

$D(B)$ καμία αλλαγή

Οπότε, το γράφημα του προβλήματος ικανοποίησης περιορισμών που εξετάσαμε, μετά την εφαρμογή της συνέπειας τόξου, έχει ως εξής:



Τα πεδία τιμών των μεταβλητών, μετά την εφαρμογή της συνέπειας τόξου, είναι πλέον $D(A) = \{1,2\},\ D(B) = \{1,2\}$ και $D(C) = \{1,2\}$. Παρατηρούμε ότι όλοι οι περιορισμοί ικανοποιούνται, επομένως το πρόβλημα ικανοποίησης περιορισμών του παραδείγματος παρουσιάζει συνέπεια τόξου.

3.4 Ορισμός Συνέπειας Μονοπατιού (Path Consistency) και Περιορισμένης Συνέπειας Μονοπατιού (Restricted Path Consistency)

Ένα ζεύγος τιμών (a_i, a_j), με το $a_i \in D(x_i)$ και $a_j \in D(x_j)$, παρουσιάζει συνέπεια μονοπατιού αν για κάθε τρίτη μεταβλητή x_k η οποία συμμετέχει σε περιορισμό με τη x_i και τη x_j, υπάρχει μία τιμή $a_k \in D(x_k)$, τέτοια ώστε η a_k να είναι support τόσο της a_i όσο και της a_j. Σε αυτήν την περίπτωση, η a_j είναι support της συνέπειας μονοπατιού της a_i στο $D(x_j)$ και η a_k είναι «μάρτυρας» της συνέπειας μονοπατιού για το ζεύγος τιμών (a_i, a_j) στο $D(x_k)$.

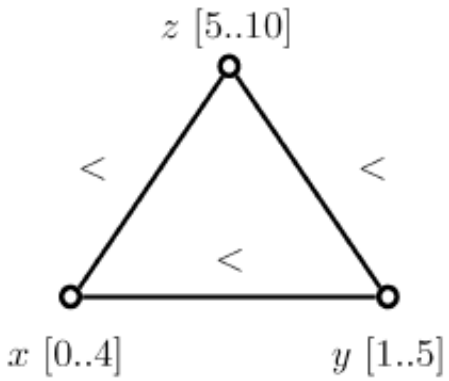
Κατ'επέκταση, μία τιμή $a_i \in D(x_i)$ παρουσιάζει περιορισμένη συνέπεια μονοπατιού (restricted path consistency – RPC) αν παρουσιάζει συνέπεια μονοπατιού και για κάθε περιορισμό c_{ij} τέτοιο ώστε η a_i έχει ένα μοναδικό support $a_j \in D(x_j)$, το ζεύγος των τιμών (a_i, a_j) παρουσιάζει συνέπεια μονοπατιού. Η έννοια της περιορισμένης συνέπειας μονοπατιού μας ενδιαφέρει για τον RPC αλγόριθμο, τον οποίο μελετούμε και υλοποιούμε σε παρακάτω κεφάλαια.

3.5 Παράδειγμα της Συνέπειας Μονοπατιού

Θα δώσουμε ένα διαισθητικό παράδειγμα για να κατανοήσουμε την έννοια της συνέπειας μονοπατιού.

Οι μεταβλητές είναι: x,y,z
Τα πεδία τιμών είναι: $D(x) = \{0,1,2,3,4\}$, $D(y) = \{1,2,3,4,5\}$ και $D(z) = \{5,6,7,8,9,10\}$
Οι περιορισμοί είναι x<y, y<z και x<z

Το γράφημα του προβλήματος έχει ως εξής:



Εξετάζουμε όλους τους περιορισμούς:

- $C_{x,y}$ = {(a,b) | a < b, a ∈ [0..4], b ∈ [1..5]} ικανοποιείται ο περιορισμός για όλες τις τιμές των μεταβλητών x και y

- $C_{y,z}$ = {(b,c) | b < c, b ∈ [1..5], c ∈ [5..10]} ικανοποιείται ο περιορισμός για όλες τις τιμές των μεταβλητών y και z

- $C_{x,z}$ = {(a,c) | a < c, a ∈ [0..4], c ∈ [5..10]} δεν ικανοποιείται ο περιορισμός

Για τις τιμές 4 στο πεδίο τιμών της μεταβλητής x και για την τιμή 5 στο πεδίο τιμών της μεταβλητής z, δεν υπάρχει τιμή στο πεδίο τιμών της μεταβλητής y, τέτοια ώστε να ισχύουν **ταυτόχρονα** οι περιορισμοί y>4 και y<5.

Επομένως, το πρόβλημα του παραδείγματος δεν παρουσιάζει συνέπεια μονοπατιού. Θα πρέπει στην πορεία να προχωρήσουμε σε διαγραφές τιμών από τα πεδία τιμών των μεταβλητών, ώστε να επιτύχουμε συνέπεια μονοπατιού.

Κεφάλαιο 4

Ο Αλγόριθμος AC-3

Στο 4^{ο} κεφάλαιο της διπλωματικής εργασίας παρουσιάζεται εκτενώς ο αλγόριθμος AC-3. Αρχικά, παρατίθεται η ανάλυση του αλγορίθμου τόσο σε επίπεδο θεωρίας όσο και σε επίπεδο υλοποίησης. Έπειτα, ακολουθεί ο ψευδοκώδικας του AC-3, ώστε να γίνει κατανοητός ο τρόπος λειτουργίας του αλγορίθμου. Στο τέλος του κεφαλαίου εξετάζουμε την χρονική και χωρική πολυπλοκότητα του αλγορίθμου και επίσης μέσω παραδείγματος διαπιστώνουμε τι είδους συνέπεια επιτυγχάνει ο AC-3.

4.1 Ανάλυση Δομής του αλγορίθμου AC-3

Ο πιο διαδεδομένος και ευρέως χρησιμοποιούμενος αλγόριθμος συνέπειας τόξου (arc consistency algorithm) δεν είναι άλλος από τον AC-3. Προτάθηκε και αναπτύχθηκε από τον καθηγητή Alan Mackworth το 1977. Οι προηγούμενοι AC αλγόριθμοι θεωρούνται ανεπαρκείς σε σημαντικό βαθμό, καθώς και αρκετά δυσκολότεροι στην εφαρμογή τους. Έτσι λοιπόν, ο αλγόριθμος AC-3 χρησιμοποιείται κατά κόρον, ενίοτε και σε συνδυασμό με άλλους αλγορίθμους όπως ο DFS ή ο BT, για την επίλυση πραγματικών προβλημάτων ικανοποίησης περιορισμών.

Ο αλγόριθμος AC-3 είναι απλοϊκός στην κατανόηση του και βρίσκει εφαρμογή σε προβλήματα ικανοποίησης δυαδικών περιορισμών, όπου πετυχαίνει συνέπεια τόξου για τόξα με δύο κορυφές. Η ειδοποιός διαφορά του με τους προγόνους του, AC-1 και AC-2, είναι ότι δε χρειάζεται να επεξεργαστούμε εξ αρχής όλους τους περιορισμούς, εάν μόνο λίγα πεδία τιμών των μεταβλητών έχουν αλλάξει. Ο αλγόριθμος AC-3 κρατά τα τόξα των μεταβλητών των οποίων τα πεδία τιμών υφίστανται αλλαγή, σε μια FIFO πολιτικής ουρά, ώστε να επεξεργαστεί αυτά μόνο.

Η ουρά Q αρχικοποιείται με όλα τα τόξα (x_k, x_i) όπου $i \neq k$, τα οποία τόξα ανήκουν στο σύνολο των περιορισμών ενός προβλήματος CSP. Έπειτα, ακολουθεί η χρήση ενός

επαναληπτικού βρόγχου, ο οποίος σε κάθε του επανάληψη αφαιρεί ένα τόξο (x_k, x_i) από την ουρά Q, ώστε να εξετασθεί για πιθανές αλλαγές στα πεδία τιμών των μεταβλητών αυτών από την συνάρτηση REVISE(x_k, x_i). Ο επαναληπτικός βρόγχος συνεχίζει την εκτέλεση του, μέχρις ότου η ουρά Q να «αδειάσει», δηλαδή να μην υπάρχουν πλέον άλλα τόξα προς αναθεώρηση από την συνάρτηση REVISE(x_k, x_i). Αν η REVISE(x_k, x_i) πραγματοποιήσει κάποια διαγραφή τιμής από το πεδίο τιμών της εκάστοτε x_k , τότε ο αλγόριθμος προσθέτει στην ουρά Q όλα εκείνα τα τόξα που δεν βρίσκονται ήδη στην ουρά και περιλαμβάνουν τη μεταβλητή x_k ως μία εκ των δύο κορυφών τους. Η ουρά Q αποθηκεύει ουσιαστικά όλα εκείνα τα τόξα τα οποία δεν είναι βέβαιο ότι παρουσιάζουν συνέπεια τόξου. Αφού η ουρά Q αδειάσει από τόξα, μπορούμε να ισχυριστούμε με βεβαιότητα, πως όλα τα πεδία τιμών των μεταβλητών ενός CSP παρουσιάζουν τοπική συνέπεια τόξου για όλους τους περιορισμούς. Εδώ έγκειται και το βασικό πλεονέκτημα του AC-3 συγκριτικά με τους προκατόχους του: Χρησιμοποιώντας την ουρά Q για τη διατήρηση των τόξων (x_k, x_i) αποφεύγεται η άσκοπη κλήση της συνάρτησης REVISE(x_k, x_i). Οι προγενέστεροι AC αλγόριθμοι δεν χρησιμοποιούσαν καμία ουρά. Συνεπώς, η χρήση της ουράς Q από τον AC-3 βελτιστοποιεί σημαντικά το χρόνο εκτέλεσης του αλγορίθμου, καθώς και «συμφέρει» προγραμματιστικά.

Βασικό συστατικό του αλγορίθμου AC-3 αποτελεί η συνάρτηση REVISE(x_k, x_i). Δέχεται ως είσοδο το εκάστοτε τόξο (x_k, x_i) και παίρνει κάθε τιμή a_k που ανήκει στο πεδίο τιμών $D(x_k)$ και ελέγχει αν υπάρχει έστω και μία τιμή a_i στο πεδίο τιμών $D(x_i)$, έτσι ώστε η τιμή a_k να παρουσιάζει συνέπεια τόξου ως προς την τιμή a_i. Αν δε βρεθεί κάποια τέτοια τιμή a_i, τότε η τιμή a_k διαγράφεται από το πεδίο τιμών $D(x_k)$. Έστω και μόλις μία αφαίρεση τιμής a_k από το πεδίο τιμών $D(x_k)$ να πραγματοποιηθεί, η συνάρτηση REVISE(x_k, x_i) επιστρέφει την τιμή true. Αν δεν διαγραφεί καμία τιμή a_k από το πεδίο τιμών $D(x_k)$, αυτό σημαίνει πως η μεταβλητή x_k παρουσιάζει συνέπεια τόξου ως προς τη x_i και η συνάρτηση REVISE(x_k, x_i) επιστρέφει false.

4.2 Ψευδοκώδικας του αλγορίθμου AC-3

Για να γίνει αντιληπτός ο αλγόριθμος AC-3 και ο τρόπος λειτουργίας του, θα παραθέσουμε τον ψευδοκώδικα τόσο του AC-3 όσο και της συνάρτησης REVISE(x_k, x_i). Να

σημειώσουμε πως (x_k, x_i) είναι τα τόξα μεταξύ των κορυφών και υποδηλώνουν τον περιορισμό μεταξύ των x_k και x_i. Τα πεδία τιμών των x_k και x_i είναι τα $D(x_k)$ και $D(x_i)$ αντίστοιχα. Η ουρά Q είναι μια FIFO ουρά που αρχικοποιείται στην αρχή του αλγορίθμου με όλα τα τόξα (x_k, x_i) του προβλήματος CSP και έπειτα διαλέγουμε και αφαιρούμε ένα προς ένα τα τόξα αυτά, ώστε να ελεγχθούν για συνέπεια τόξου από την συνάρτηση REVISE(arc(x_i, x_j)). Ως arcs(G) ορίζουμε το σύνολο των τόξων-περιορισμών του γραφήματος G. Η μεταβλητή removed στην συνάρτηση REVISE(arc(x_i, x_j)) είναι τύπου Boolean και αν γίνει έστω και μία διαγραφή τιμής από κάποιο πεδίο τιμών μιας μεταβλητής επιστρέφει true, σε κάθε άλλη περίπτωση επιστρέφει false. Μόλις η ουρά Q αδειάσει από τόξα, τερματίζεται η εκτέλεση του αλγορίθμου AC-3.

```
Algorithm AC-3 {
Q ={ (x_k, x_i)in arcs(G),k<>i }; // Q initialization
while ( !empty(Q) ) {
    arc((x_i, x_j)) = pop( Q ); // select and delete first arc (x_i, x_j)from Q;
            if(REVISE(arc(x_i, x_j))) == true ) {
            // if not arc consistent, then add arcs ((x_m, x_i)) to the Q…
            Q <- Q union {(x_m, x_i) such that (x_m, x_i)in arcs(G), i≠j, i≠m}
            endif;
 endwhile;
        } //end if condition
    } // end while loop
} // end AC-3 Algorithm
```

```
function REVISE(arc(x_i, x_j))  {
removed = false;
```

```
    for each a_i in D((x_i))
        if (no x_j exists such that (a_i, a_j) is arc-consistent)
            remove a_i from D((x_i));
            removed = true;
        endif;
    endfor;
return removed; //
end REVISE(arc((x_i, x_j))
```

4.3 Χρονική και Χωρική Πολυπλοκότητα του AC-3

4.3.1 Χρονική Πολυπλοκότητα του AC-3

Όπως γνωρίζουμε πλέον, ο AC-3 διατηρεί μια fifo ουρά Q, η οποία αποθηκεύει όλα τα τόξα των περιορισμών που συμμετέχουν στο πρόβλημα δυαδικών περιορισμών. Έστω λοιπόν ένας περιορισμός C_{ij}. Για να εισαχθεί εκ νέου στην Q ο εν λόγω περιορισμός θα πρέπει το πεδίο τιμών ενός εκ των δύο μεταβλητών που συμμετέχουν στον περιορισμό να έχει αλλάξει, δηλαδή να έχει πραγματοποιηθεί τουλάχιστον μία διαγραφή τιμής σε κάποιο από τα πεδία τιμών. Υπάρχουν το πολύ $2 * d$ τιμές, δηλαδή d τιμές για κάθε μία εκ των δύο μεταβλητών. Αυτό πρακτικά σημαίνει ότι ο αλγόριθμος AC-3 θα επεξεργαστεί τον περιορισμό C_{ij}το πολύ $2 * d$ φορές. Οι περιορισμοί είναι συνολικά e και η επεξεργασία του καθενός απαιτεί χρόνο ίσο με $O(d^2)$, συνεπώς η χρονική πολυπλοκότητα του AC-3 είναι $O(e * d^3)$.

Στο σημείο αυτό να σημειώσουμε ότι για προβλήματα περιορισμών τα οποία παρουσιάζουν συνέπεια τόξου εξ αρχής, ο αλγόριθμος απαιτεί χρόνο ίσο με $O(e * d^2)$, διότι κάθε περιορισμός C_{ij} θα επεξεργασθεί από τη συνάρτηση revise μόλις μία φορά και δε θα εισαχθεί στην Q εκ νέου, καθώς δε θα γίνει καμία τροποποίηση στα πεδία τιμών του.

Η χρονική πολυπλοκότητα του AC-3 δεν είναι όμως βέλτιστη. Η χρήση της ουράς Q σε συνδυασμό με την συνάρτηση revise έχει ως αποτέλεσμα να μην ελέγχονται ξανά οι περιορισμοί για τους οποίους γνωρίζουμε ότι δεν έχει γίνει καμία αλλαγή στα πεδία τιμών των μεταβλητών τους και συνεπώς παρουσιάζουν ήδη συνέπεια τόξου. Σαφώς και είναι επιθυμητό το τελευταίο. Ωστόσο, πρέπει να θυμόμαστε ότι η Revise δεν αποθηκεύει απολύτως κανένα στοιχείο για τους υπολογισμούς που κάνει με σκοπό να βρει support για τις τιμές των μεταβλητών. Το γεγονός αυτό, αναγκάζει τον AC-3 να ξανακάνει αρκετές φορές ελέγχους μεταξύ των τιμών των μεταβλητών για τις οποίες έχει ήδη κάνει έλεγχο για συνέπεια τόξου.

4.3.2 Χωρική Πολυπλοκότητα του AC-3

Η χωρική πολυπλοκότητα του AC-3 έγκειται στην μόνη δομή δεδομένων Q που χρησιμοποιεί. Η ουρά Q κρατάει τα τόξα-περιορισμούς που πρέπει να ελεγχθούν για συνέπεια τόξου. Αρχικά, διατηρούνται όλα τα τόξα στην Q, τα οποία ένα-ένα αφαιρούνται και προωθούνται για έλεγχο συνέπειας τόξου από τη revise και αν πραγματοποιηθεί κάποια αλλαγή στα πεδία τιμών των μεταβλητών τότε μόνο προστίθενται στην Q ξανά. Με την προϋπόθεση ότι τα τόξα που υπάρχουν ήδη στην ουρά δεν προστίθενται εκ νέου, είναι βέβαιο ότι τα τόξα που είναι αποθηκευμένα στην ουρά δεν θα ξεπεράσουν ποτέ σε αριθμό τον συνολικό αριθμό των περιορισμών του προβλήματος που είναι ίσος με e. Συνεπώς, η χωρική πολυπλοκότητα είναι $O(e)$.

4.4 Τοπική συνέπεια και ο αλγόριθμος AC-3

Ο AC-3 επιτυγχάνει μόνο τοπική συνέπεια, δηλαδή συνέπεια τόξου μιας μεταβλητής με μία άλλη, σε σχέση με τον δυαδικό περιορισμό που τις συνδέει. Δεν πετυχαίνει ολική ή παγκόσμια συνέπεια τόξου(global arc consistency) για ολόκληρο το CSP που εξετάζει.

Για να το αντιληφθούμε καλύτερα αυτό, έστω ότι έχουμε το ακόλουθο CSP με τις μεταβλητές:

- X, Y και Z.

- Η X έχει πεδίο τιμών το (1 2 3),
- Η Y και η Z έχουν πεδίο τιμών το (1 2)

Οι δυαδικοί περιορισμοί είναι:

- $X \neq Y$
- $Y \neq Z$
- $X \neq Z$

Εφαρμόζοντας μόνο τον AC-3 αλγόριθμο, καμία διαγραφή τιμής δε θα πραγματοποιηθεί από κανένα πεδίο τιμών κάποιας μεταβλητής, το οποίο σημαίνει ότι παρουσιάζουν συνέπεια τόξου.

Αυτό όμως δεν σημαίνει ότι οι μεταβλητές X, Y και Z μπορούν να πάρουν όποια τιμή από τα πεδία τιμών τους θέλουν, γιατί αν $X=1$ ή αν $X=2$, δεν υπάρχει λύση για το πρόβλημα.

Πρακτικά αυτό σημαίνει ότι οι τιμές 1 και 2 από το πεδίο τιμών της μεταβλητής X δεν παρουσιάζουν ολική συνέπεια τόξου (globally arc-consistent) , διότι αυτές οι δύο τιμές δεν οδηγούν σε λύση.

Αλγόριθμοι όπως ο AC-3 δεν είναι σε θέση να ανιχνεύσουν τέτοιου είδους ασυνέπειες. Φυσικά υπάρχουν τεχνικές και άλλοι αλγόριθμοι που το επιτυγχάνουν, ωστόσο είναι πολύ δαπανηροί υπολογιστικά και δε θα τους εξετάσουμε στην παρούσα διπλωματική εργασία.

Κεφάλαιο 5

Ο Αλγόριθμος RPC

Στο 5^{ο} κεφάλαιο της διπλωματικής εργασίας παρουσιάζεται εκτενώς ο αλγόριθμος RPC(Restricted Path Consistency). Αρχικά, παρατίθεται η ανάλυση του αλγορίθμου RPC, καθώς και του τρόπου λειτουργίας του. Έπειτα, παρουσιάζεται η χωρική και χρονική του πολυπλοκότητα. Στο τέλος του κεφαλαίου 5 ακολουθεί ο ψευδοκώδικας του RPC αλγορίθμου, ώστε να γίνει κατανοητός ο τρόπος λειτουργίας του αλγορίθμου.

5.1 Ανάλυση Δομής του αλγορίθμου RPC

Ο αλγόριθμος RPC που υλοποιείται στην παρούσα εργασία, αποτελεί προγραμματιστική προέκταση του αλγορίθμου AC-3 που αναλύσαμε σε προηγούμενο κεφάλαιο. Για την ακρίβεια, μελετούμε και υλοποιούμε την πιο «ελαφριά έκδοση» του συγκεκριμένου αλγορίθμου, τον RPC3, χωρίς τις δομές R_1 και R_2, οι οποίες χρησιμοποιούνται για την καταγραφή και αποθήκευση «υπολοίπων» από τους υπολογισμούς που εκτελούνται. Για λόγους ευκολίας, θα αναφερόμαστε στον αλγόριθμο αυτόν ως RPC. Οι υπολογισμοί που πραγματοποιεί είναι συνήθως πιο χρονοβόροι σε σχέση με τους προκατόχους του (RPC1, RPC2) με αποτέλεσμα να αυξάνεται η χρονική του πολυπλοκότητα. Ωστόσο, ο αλγόριθμος και οι δομές που χρησιμοποιεί δεν είναι πιο ακριβές σε υπολογιστικούς πόρους και ως εκ τούτου η χωρική του πολυπλοκότητα συγκριτικά με τους προγενέστερους RPC αλγορίθμους είναι μικρότερη. Ο αλγόριθμος RPC είναι σχετικά απλός στην κατανόηση του και βρίσκει εφαρμογή σε προβλήματα ικανοποίησης δυαδικών περιορισμών, όπου πετυχαίνει περιορισμένη συνέπεια μονοπατιού για τόξα με δύο κορυφές. Σε αυτό το σημείο να υπενθυμιστεί ο ορισμός της περιορισμένης συνέπειας μονοπατιού, ώστε να γίνει κατανοητή η μορφής της συνέπειας που επιτυγχάνει ο αλγόριθμος. Ορίζουμε ότι μία τιμή $a_i \in D(x_i)$ παρουσιάζει περιορισμένη συνέπεια μονοπατιού (restricted

path consistency – RPC) αν παρουσιάζει συνέπεια μονοπατιού και για κάθε περιορισμό c_{ij} τέτοιο ώστε η a_i έχει ένα μοναδικό support $a_j \in D(x_j)$, το ζεύγος των τιμών (a_i, a_j) παρουσιάζει συνέπεια μονοπατιού.

Ο αλγόριθμος RPC, όπως έχει αναφερθεί ήδη, αποτελεί προγραμματιστική προέκταση του AC-3 και αυτό γίνεται κατανοητό από τις δομές του και τον τρόπο λειτουργίας αυτών. Όπως και ο AC-3, δημιουργεί και διατηρεί μία FIFO πολιτικής ουρά, την Q, η οποία αρχικοποιείται με όλα τα τόξα (x_ι, x_j) , με $i \neq j$, τα οποία τόξα ανήκουν στο σύνολο των δυαδικών περιορισμών ενός προβλήματος CSP. Έπειτα, ακολουθεί η χρήση ενός επαναληπτικού βρόγχου, ο οποίος σε κάθε του επανάληψη αφαιρεί ένα τόξο (x_i, x_j) από την ουρά Q, ώστε να εξετασθεί από την συνάρτηση FindTwoSupports(x_i, x_j). Η συνάρτηση FindTwoSupports(x_i, x_j) αποτελεί, τρόπον τινά, παραλλαγή και προέκταση της Revise συνάρτησης που χρησιμοποιεί ο AC-3. Όπως ίσως υποδηλώνει και το όνομα της, ο σκοπός της είναι να βρει κανένα, ένα έως και δύο τιμές support για την τιμή a_i στο πεδίο τιμών $D(x_j)$. Εφόσον βρεθεί μία τέτοια τιμή support $a_j \in D(x_j)$ ως το μοναδικό support για την τιμή a_i, τότε γίνεται έλεγχος αν το ζεύγος (a_i, a_j) παρουσιάζει συνέπεια μονοπατιού (path consistency). Αν δεν βρεθεί καμία τέτοια τιμή, η συνάρτηση προχωρά σε διαγραφή της τιμής a_i από το $D(x_i)$.

Τον έλεγχο συνέπειας μονοπατιού πραγματοποιεί η συνάρτηση PC_search(a_i, a_j, x_k), η οποία καλείται μέσα από την FindTwoSupports(x_i, x_j). Αρχικά, η συνάρτηση αναζητεί και αποθηκεύει σε μία δομή, όλες τις μεταβλητές x_k που συμμετέχουν από κοινού σε περιορισμό και με την x_i και με την x_j, με την προϋπόθεση ότι $x_k \neq x_i$ και $x_k \neq x_j$ και φυσικά $(x_i, x_k) \in C$, $(x_j, x_k) \in C$. Στην συνέχεια, ελέγχει μέσα σε ένα βρόγχο επανάληψης για όλες τις τιμές $a_k \in D(x_k)$ αν τα ζεύγη τιμών (a_i, a_κ) και (a_j, a_κ) ικανοποιούν τους περιορισμούς του προβλήματος. Αν βρεθεί μια τέτοια τιμή a_κ η οποία ονομάζεται pc-witness για το ζευγάρι (a_i, a_j), τότε ο βρόγχος συνεχίζει με την επόμενη x_k που συμμετέχει από κοινού σε περιορισμό με την x_i και με την x_j. Ενδέχεται οι pc-witnesses τιμές να είναι πολλές, δεν ενδιαφέρει, όμως, το ποιες είναι. Το μόνο ζητούμενο της συνάρτησης PC_search(a_i, a_j, x_k) είναι αν θα επιστρέψει true, δηλαδή ότι βρέθηκε τουλάχιστον για κάθε x_k μεταβλητή από μία a_κ τιμή ως pc-witness για το ζευγάρι (a_i, a_j), αλλιώς θα επιστρέψει false. Αν δεν βρεθεί σε καμία x_k κάποια τιμή pc-witness, η συνάρτηση PC_search(a_i, a_j, x_k) επιστρέφει false, γεγονός που σημαίνει ότι η τιμή a_i δεν είναι RPC, ή με άλλα λόγια πως η τιμή a_i δεν παρουσιάζει περιορισμένη συνέπεια

μονοπατιού και πρέπει αυτή η τιμή a_i να διαγραφεί από το $D(x_i)$. Η ροή εκτέλεσης του αλγορίθμου μετά την PC_search(a_i, a_j, x_k) επιστρέφει στην συνάρτηση FindTwoSupports(x_i, x_j), όπου γίνονται οι σχετικές διαγραφές τιμών ή όχι, και η τελευταία με τη σειρά της επιστρέφει την ροή εκτέλεσης στον κύριο κώδικα του RPC, για την εξέταση του επόμενου τόξου (x_i, x_j) που βρίσκεται πρώτο στην ουρά Q.

Στην περίπτωση που πραγματοποιηθεί έστω και μία διαγραφή τιμής από κάποιο πεδίο τιμών των μεταβλητών του προβλήματος μέσα από την συνάρτηση FindTwoSupports(x_i, x_j), αυτό σημαίνει ότι πρέπει να εισαχθούν εκ νέου στο τέλος της ουράς Q όλα τα τόξα (x_m, x_i), με m≠i, ώστε να γίνει η διάδοση των περιορισμών, καθώς κάποια τόξα ενδέχεται να απολέσουν την συνέπεια μεταξύ τους. Ο τρόπος λειτουργίας της ουράς Q είναι πανομοιότυπος με τον αλγόριθμο AC-3. Ο RPC, λόγω των αυστηρότερων ελέγχων του, αναμένεται να προβαίνει σε περισσότερες διαγραφές τιμών από τον AC-3 και τερματίζει μόλις η ουρά Q έχει αδειάσει τελείως από τόξα (x_i, x_j).

5.2 Ψευδοκώδικας του αλγορίθμου RPC

Για να γίνει αντιληπτός ο αλγόριθμος RPC και ο τρόπος λειτουργίας του, θα παραθέσουμε τον ψευδοκώδικα τόσο του RPC όσο και των δύο συναρτήσεων FindTwoSupports(x_i, x_j) και PC_search(a_i, a_j, x_k). Σημειώνουμε πως (x_i, x_j) είναι τα τόξα μεταξύ των κορυφών και υποδηλώνουν τον περιορισμό μεταξύ των x_i και x_i. Τα πεδία τιμών των x_i, x_j και x_k είναι τα D(x_i), D(x_j),και D(x_k) αντίστοιχα. Η ουρά Q είναι μια FIFO ουρά που αρχικοποιείται στην αρχή του αλγορίθμου με όλα τα τόξα (x_i, x_j) του προβλήματος CSP και έπειτα αφαιρούνται ένα προς ένα τα τόξα αυτά, ώστε να ελεγχθεί αν έχουν κανένα, ένα ή δύο supports από την συνάρτηση FindTwoSupports(x_i, x_j). Ως arcs(G) ορίζουμε το σύνολο των τόξων-περιορισμών του γραφήματος G. Στην FindTwoSupports(x_i, x_j) χρησιμοποιούμε την μεταβλητή support_for_PC η οποία λαμβάνει τις τιμές 0,1 ή 2. Στην συνάρτηση PC_search(a_i, a_j, x_k) η δομή X_k αποθηκεύει τις μεταβλητές x_k που συμμετέχουν από κοινού σε περιορισμό και με την x_i και με την x_j. Μόλις η ουρά Q αδειάσει από τόξα, τερματίζεται η εκτέλεση του αλγορίθμου RPC.

Ο ψευδοκώδικας του αλγορίθμου RPC δίνεται παρακάτω:

```
Algorithm RPC{
Q ={(x_i, x_j)in arcs(G),j≠i }; // Q initialization
while ( !empty(Q) ) {
    arc((x_i, x_j)) = pop( Q ); // select and delete first arc (x_i, x_j)from Q;
            FindTwoSupports((x_i, x_j)); //call function FindTwoSupports
            if(FindTwoSupports((x_i, x_j)) == false ) {
            // if not consistent, then add arcs ((x_m, x_i)) to the Q…
            Q <- Q union {(x_m, x_i) such that (x_m, x_i)in arcs(G), i≠j, i≠m}
            endif;
 endwhile;
        } //end if condition
    } // end while loop
} // end RPC Algorithm
```

Έπειτα ακολουθεί η συνάρτηση FindTwoSupports(x_i, x_j):

```
function FindTwoSupports(x_i, x_j){
support_for_pc = false;
      for each a_j in D(x_j)
            if isConsistent(a_i, a_j) then
               if support_for_pc==false
                  support_for_pc=true;   //found support
                  PC_search(a_i, a_j, x_k);  // call PC_search
```

```
            endif;
        else                                    //if not consistent (a_i, a_j)
            remove a_i from D((x_i));
            removed = true;
          endif;
          if PC_search(a_i, a_j, x_k)==TRUE
                print("ai value is RPC"); //a_i value is RPC
                break; //proceed with next a_i value
          else
                print("ai value is not RPC"); //a_i value is not RPC
                remove a_i from D((x_i));    //delete a_i value
               removed = true;
    endfor;
return removed; //
end FindTwoSupports(x_i, x_j)
```

Η συνάρτηση PC_search(a_i, a_j, x_k) που καλείται μέσα από την FindTwoSupports(x_i, x_j) έχει ως εξής:

```
function PC_search(a_i, a_j, x_k) {
pc_witness = false;
    for each a_k in D(x_k)
        if (isConsistent(a_k, a_i) and isConsistent(a_j, a_k)) then
            pc_witness =true;   //found pc-witness for at least one value a_k of this x_k
```

```
            break;                //proceed to next a_k value
        if   pc_witness=false then
            return FALSE;  // no path consistency witness found
        endif;
    endfor;
return TRUE;
end PC_search(a_i, a_j, x_k)
```

5.3 Χρονική και Χωρική Πολυπλοκότητα του RPC

5.3.1 Χρονική Πολυπλοκότητα του RPC

Όπως ήδη γνωρίζουμε για προβλήματα περιορισμών, τα οποία παρουσιάζουν συνέπεια τόξου εξ αρχής, ο αλγόριθμος τουλάχιστον απαιτεί χρόνο ίσο με $O(e * d^2)$, διότι κάθε περιορισμός C_{ij} θα επεξεργασθεί από τη συνάρτηση FindTwoSupports μόλις μία φορά και δε θα εισαχθεί στην Q εκ νέου, καθώς δε θα γίνει καμία τροποποίηση στα πεδία τιμών του. Οι περιορισμοί είναι συνολικά e και η επεξεργασία του καθενός απαιτεί χρόνο ίσο με $O(d^2)$, συνεπώς για τον RPC μπορούμε εύκολα να καταλήξουμε ότι η χειρότερη χρονική πολυπλοκότητα του είναι $O(ned^3)$.

5.3.2 Χωρική Πολυπλοκότητα του RPC

Η χωρική πολυπλοκότητα καθορίζεται από το χώρο που απαιτείται για να αποθηκευτούν οι δομές που χρησιμοποιεί ο αλγόριθμος για την αποθήκευση και την επεξεργασία των δεδομένων του. Όντας «οικονομικότερος» σε σχέση με διάφορους πρόγονους αλγορίθμους, ο RPC δεν χρειάζεται να χρησιμοποιήσει ακριβές δομές για την αποθήκευση δεδομένων, ακόμα και σε περιπτώσεις που γίνονται αρκετές τροποποιήσεις στα δεδομένα του προβλήματος. Αυτό μας οδηγεί στο συμπέρασμα ότι η χωρική του πολυπλεξία είναι μόλις $O(ed)$.

Κεφάλαιο 6

Προγραμματιστική Υλοποίηση

Στο 6^{ο} κεφάλαιο παρουσιάζεται η προγραμματιστική υλοποίηση των αλγορίθμων AC-3 και RPC που μελετήσαμε θεωρητικά σε προηγούμενες ενότητες. Πιο συγκεκριμένα, παρατίθενται και εξετάζονται οι δομές δεδομένων που χρησιμοποιήθηκαν, συνοδευμένες από επεξηγηματικά σχόλια. Ακολουθούν διάφορα στιγμιότυπα της ροής του κώδικα, συνοδευμένα και αυτά από σχόλια, καθώς και τα αποτελέσματα από διάφορες εκτελέσεις των αλγορίθμων. Στο τέλος του παρόντος κεφαλαίου, θα παρουσιαστούν σε ένα συγκεντρωτικό πίνακα τα στατιστικά στοιχεία εκτέλεσης των αλγορίθμων.

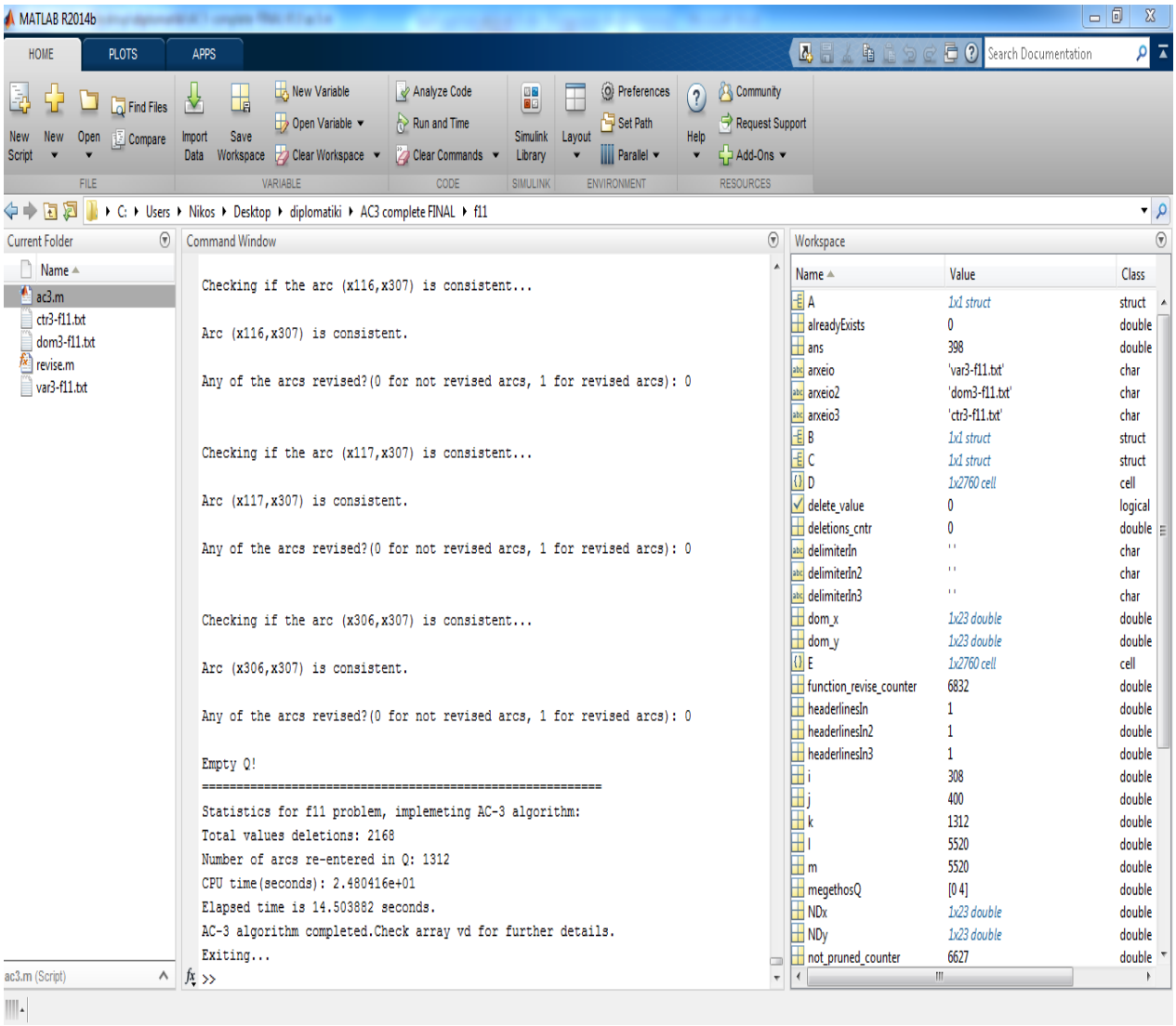
6.1 Γενική Περιγραφή Λειτουργίας Προγράμματος

Στην εν λόγω διπλωματική εργασία, οι αλγόριθμοι AC-3 και RPC υλοποιήθηκαν προγραμματιστικά με τη χρήση της γλώσσας προγραμματισμού MATLAB (έκδοση 2014b).

Το πρόγραμμα λαμβάνει 3 αρχεία εισόδου, με κατάλληλη μορφοποίηση, αποθηκευμένα ως αρχεία .txt. Κάθε πρόβλημα που εισάγουμε στο πρόγραμμα μας, αποτελείται από τα αρχεία var, dom και con. Τα αρχεία var περιέχουν τον συνολικό αριθμό των μεταβλητών του προβλήματος, καθώς και από ποιο πεδίο ορισμού παίρνουν τιμές αυτές οι μεταβλητές. Τα αρχεία dom περιέχουν το σύνολο των πεδίων ορισμού, το σύνολο των τιμών για κάθε πεδίο ορισμού. Τέλος, τα αρχεία con περιέχουν το σύνολο των περιορισμών του προβλήματος και τις μεταβλητές που συμμετέχουν σε κάθε περιορισμό. Αυτά τα .txt αρχεία πρέπει να βρίσκονται ήδη μέσα στον φάκελο που περιέχει τα αρχεία κώδικα του προγράμματος, πριν την έναρξη της εκτέλεσης.Το πρόγραμμα διαβάζει τα δεδομένα του προβλήματος από τα αρχεία .txt και τα αποθηκεύει σε εσωτερικές δομές δεδομένων, όπως θα δούμε και παρακάτω.

Ύστερα από το πέρας της εκτέλεσης του κάθε αλγορίθμου, παρουσιάζονται τα αποτελέσματα και τα στατιστικά στοιχεία για το κάθε εισαγόμενο CSP πρόβλημα. Τα μετρικά

αυτά μεγέθη είναι οι συνολικές διαγραφές τιμών που πραγματοποίησε ο αλγόριθμος από τα πεδία τιμών των μεταβλητών, το πλήθος των τόξων που εισήχθησαν εκ νέου στην ουρά Q προς επανεξέταση, ο χρόνος CPU σε δευτερόλεπτα και ο χρόνος εκτέλεσης επίσης σε δευτερόλεπτα. Επίσης, στο workspace, μπορούμε να έχουμε πρόσβαση σε όλες τις μεταβλητές και δομές δεδομένων του προβλήματος, ώστε να εξετάσουμε λεπτομερώς οτιδήποτε μας ενδιαφέρει. Για παράδειγμα:

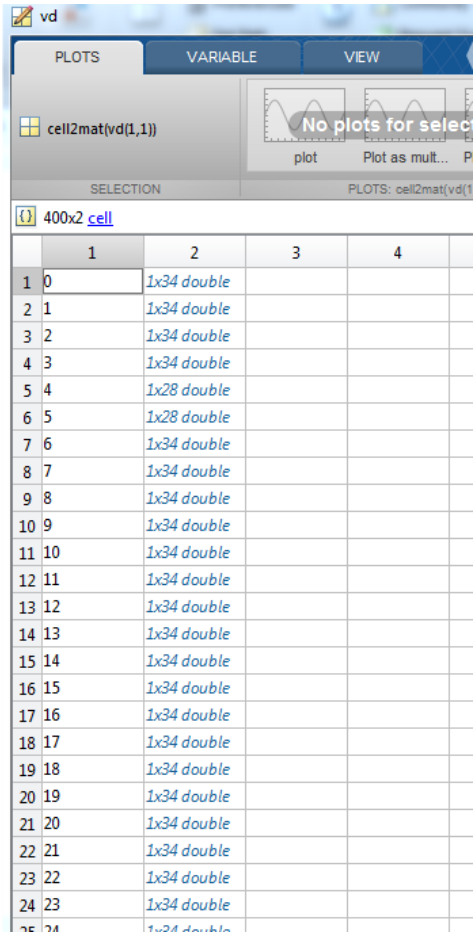


Εικόνα6.1.1~ *Στιγμιότυπο μετά την εκτέλεση του AC-3 αλγορίθμου για το f11 πρόβλημα. Στο κέντρο της οθόνης εμφανίζονται τα στατιστικά μεγέθη του προγράμματος και στα δεξιά, στο workspace, παρατίθενται όλες οι μεταβλητές και δομές δεδομένων που χρησιμοποιήθηκαν.*

6.2 Οι Δομές Δεδομένων των Αλγορίθμων

Ο κώδικας και στους δύο αλγορίθμους χρησιμοποιεί συγκεκριμένες δομές δεδομένων καθώς και ένα μεγάλο πλήθος από μεταβλητές, ώστε να ολοκληρωθεί με επιτυχία και να συλλέξουμε τα κατάλληλα στατιστικά μεγέθη που μας ενδιαφέρουν. Θα προχωρήσουμε σε επιμέρους ανάλυση των σημαντικότερων εξ' αυτών των δομών:

- Η δομή **vd**, που τα αρχικά της σημαίνουν variable-domain, είναι ένας cell-array ο οποίος αποθηκεύει τις μεταβλητές του εκάστοτε προβλήματος στην πρώτη στήλη και στη δεύτερη στήλη αποθηκεύει όλα τα αντίστοιχα domains, δηλαδή τα πεδία τιμών των μεταβλητών του προβλήματος. Είναι μία ιδιαίτερα χρήσιμη δομή και χρησιμοποιείται και από τους δύο αλγορίθμους σε μεγάλο βαθμό:

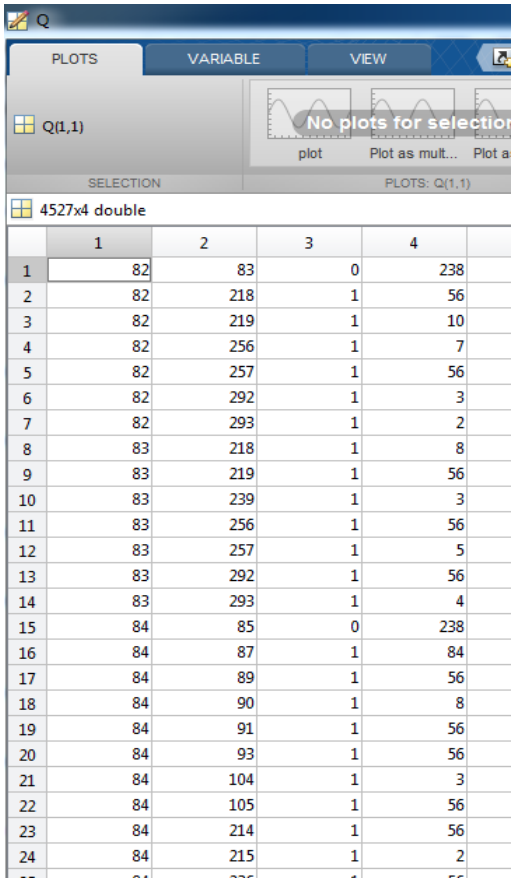


	1	2	3	4
1	0	1x34 double		
2	1	1x34 double		
3	2	1x34 double		
4	3	1x34 double		
5	4	1x28 double		
6	5	1x28 double		
7	6	1x34 double		
8	7	1x34 double		
9	8	1x34 double		
10	9	1x34 double		
11	10	1x34 double		
12	11	1x34 double		
13	12	1x34 double		
14	13	1x34 double		
15	14	1x34 double		
16	15	1x34 double		
17	16	1x34 double		
18	17	1x34 double		
19	18	1x34 double		
20	19	1x34 double		
21	20	1x34 double		
22	21	1x34 double		
23	22	1x34 double		
24	23	1x34 double		

Εικόνα6.2.1 *Η δομή vd*

Αυτό σημαίνει, για παράδειγμα, ότι η μεταβλητή x0, έχει πεδίο τιμών με 34 values, η μεταβλητή x1 το ίδιο κ.ο.κ. Η δομή vd δεν έχει σταθερό μέγεθος, αλλά το μέγεθος της καθορίζεται από το πλήθος των μεταβλητών του εισαγόμενου CSP.

- Η δομή **Q**, ίσως η σημαντικότερη δομή δεδομένων των αλγορίθμων AC-3 και RPC που μελετούμε και υλοποιούμε στην παρούσα διπλωματική εργασία. Είναι η δομή που αποθηκεύει όλα τα τόξα των περιορισμών του εκάστοτε CSP, διαγράφει το πρώτο τόξο σε κάθε loop επανάληψης, προβαίνει σε εισαγωγή ή μη τόξων στην ουρά προς επανέλεγχο σε περιπτώσεις διαγραφών τιμών από πεδία τιμών μεταβλητών και τέλος, μόλις αυτή η δομή αδειάσει, δηλαδή δεν υπάρχουν άλλα τόξα προς εξέταση από τις συναρτήσεις Revise και FindTwoSupports για τον AC-3 και RPC αντίστοιχα, τότε μόνο τερματίζει η εκτέλεση του προγράμματος. Ο τύπος της είναι double και το μέγεθος του con αρχείου, δηλαδή το πλήθος των περιορισμών μεταξύ των μεταβλητών, καθορίζουν και το μέγεθος της Q.

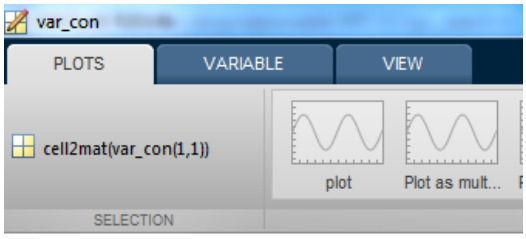


	1	2	3	4
1	82	83	0	238
2	82	218	1	56
3	82	219	1	10
4	82	256	1	7
5	82	257	1	56
6	82	292	1	3
7	82	293	1	2
8	83	218	1	8
9	83	219	1	56
10	83	239	1	3
11	83	256	1	56
12	83	257	1	5
13	83	292	1	56
14	83	293	1	4
15	84	85	0	238
16	84	87	1	84
17	84	89	1	56
18	84	90	1	8
19	84	91	1	56
20	84	93	1	56
21	84	104	1	3
22	84	105	1	56
23	84	214	1	56
24	84	215	1	2

Εικόνα6.2.2*~Η δομή Q*

Στο παραπάνω στιγμιότυπο, η $1^{η}$ γραμμή αντιστοιχεί στον εξής περιορισμό, όπως αυτός προέρχεται από το con αρχείο που έχουμε ήδη εισάγει: $|x82 - x83 = 238|$, δηλαδή απόλυτη τιμή της πρώτης μεταβλητής μείον τη δεύτερη ίσο με μία σταθερά. Έχουμε ορίσει αυθαίρετα ότι το 0 στην Τρίτη στήλη υποδηλώνει ισότητα, το 1 ανισότητα και λοιπά. Γενικότερα, στην πρώτη στήλη της Q αποθηκεύεται η πρώτη μεταβλητή που συμμετέχει στον περιορισμό, στη δεύτερη στήλη η δεύτερη μεταβλητή που συμμετέχει στον περιορισμό, στην τρίτη στήλη ο περιορισμός που μπορεί να είναι ανισοτικός, ισοτικός, συγκριτικός κλπ, και στην τέταρτη στήλη η σταθερά (εφόσον υπάρχει) με την οποία συγκρίνονται οι μεταβλητές του εκάστοτε περιορισμού. Όλες οι τιμές είναι τύπου double.

- Η δομή **var_con**. Είναι μία πολύ χρήσιμη δομή που είναι χρήσιμη αποκλειστικά για τον RPC αλγόριθμο. Αποτελεί μία δομή η οποία αναζητεί και αποθηκεύει όλες τις μοναδικές μεταβλητές που συμμετέχουν σε περιορισμό μεταξύ τους. Είναι μείζονος σημασίας καθώς αποτελεί τη βάση για την εύρεση των κοινών xk μεταβλητών που είδαμε ότι χρειάζεται οπωσδήποτε να γνωρίζει ο RPC προκειμένου να επιτύχει συνέπεια μονοπατιού για δύο μεταβλητές, μέσω της συνάρτησης pc_search. Για παράδειγμα, έχουμε:



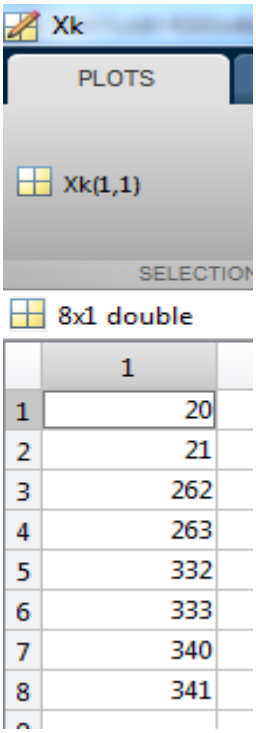
371x2 cell

	1	2
1	0	*12x1 double*
2	1	*11x1 double*
3	2	*11x1 double*
4	3	[126;128;216;217;252;253;302;303;342;343]
5	4	[5;8;9;196;197;198;199;384;385]
6	5	[8;9;196;197;198;199;384;385]
7	6	[7;8;9;10;11;26;27;384;385]
8	7	[8;9;10;11;26;27;384;385]
9	8	[9;10;11;384;385;392;393;394;395]
10	9	[10;11;384;385;392;393;394;395]
11	10	[11;16;17;392;393]
12	11	[392;393]
13	12	[13;14;15;294;295;380;381;390;391]
14	13	[14;15;294;295;380;381;390;391]
15	14	[15;294;295;380;381;390;391]

Εικόνα6.2.3*~Η δομή var_con*

Για να επεξηγήσουμε τη δομή, βασιζόμενοι στο στιγμιότυπο που παραθέτουμε, έχουμε ότι η μεταβλητή x0 συμμετέχει σε περιορισμό με άλλες 12 μεταβλητές (λόγου πλήθους δε φαίνονται, τις βλέπουμε κάνοντας κλικ εκεί), η x1 με 11 μεταβλητές, η x2 με 11 μεταβλητές επίσης, η x3 συμμετέχει σε περιορισμό με τις μεταβλητές x126,x128,x216,x217,x252,x253, x302, x303, x342 και x343 κ.ο.κ. για όλες τις υπόλοιπες μεταβλητές ανάλογα με το κάθε εισαγόμενο CSP πρόβλημα.

- Η δομή **Xk**. Λειτουργώντας ως προέκταση της προηγούμενης δομής var_con, η δομή Xk αποθηκεύει όλες τις μοναδικές τιμές των μεταβλητών που συμμετέχουν σε περιορισμό για κάθε ξεχωριστό ζεύγος (x_i, x_j) που εξετάζονται για συνέπεια μονοπατιού από την συνάρτηση pc_search. Σχηματίζεται στην συνάρτηση pc_search. Λόγου χάρη, στο f11 πρόβλημα, για τις μεταβλητές $x_i = 22$ και $x_j = 23$ η δομή έχει ως εξής:



	1
1	20
2	21
3	262
4	263
5	332
6	333
7	340
8	341

Εικόνα6.2.4~*Η δομή Xk*

Η δομή Xk, στο εν λόγω στιγμιότυπο, μας δηλώνει ότι οι κοινές μεταβλητές με τις οποίες αμφότερες οι x22 και x23 συμμετέχουν σε περιορισμό είναι οι x20, x21, x262, x263,

x332, x333, x340 και x341. Μέσα στα πεδία τιμών των συγκεκριμένων μεταβλητών θα προσπαθήσει η pc_search να βρει αν υφίσταται ή όχι συνέπεια μονοπατιού για το ζεύγος (x22,x23).

6.3 Ανάλυση Υλοποίησης του Κώδικα

Ο κώδικας των αλγορίθμων AC-3 και RPC είναι χωρισμένος σε 3 βασικές ενότητες. Η πρώτη ενότητα είναι το διάβασμα των εξωτερικών .txt αρχείων του κάθε CSP και η αποθήκευση αυτών σε κατάλληλες δομές. Για παράδειγμα:

```
%%%%%%%%%%%%%%%%DIAVASMA EKSWTERIKWN ARXEIWN%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%var file
t=cputime;  %enarksi cputime
tic;
fprintf('\nReading var3-f10.txt file...\n');
arxeio='var3-f10.txt';
delimiterIn=' ';
headerlinesIn=1;
A=importdata(arxeio,delimiterIn,headerlinesIn);
%dom file
fprintf('\nReading dom3-f10.txt file...\n');
arxeio2='dom3-f10.txt';
delimiterIn2=' ';
headerlinesIn2=1;
B=importdata(arxeio2,delimiterIn2,headerlinesIn2);
%con file
fprintf('\nReading ctr3-f10.txt file...\n');
arxeio3='ctr3-f10.txt';
delimiterIn3=' ';
headerlinesIn3=1;
C=importdata(arxeio3,delimiterIn3,headerlinesIn3);
for i=2:length(C.textdata)
    C.textdata{i,4}=C.data(i-1);
end
C.textdata(1,:)=[];
%%%%%%%%%%%%%%%%TELOS DIAVASMATOS EKSWTERIKWN ARXEIWN%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Εικόνα6.3.1~*Το διάβασμα των εξωτερικών αρχείων .txt και η αποθήκευση τους σε δομές*

Έπειτα, ακολουθεί η δημιουργία των πινάκων και δομών των αλγορίθμων. Ειδικότερα, δημιουργείται η δομή vd που αναλύσαμε παραπάνω, η δομή Q ως ένωση επιμέρους μικρότερων

δομών και γίνεται η μετατροπή των ανισο-ισοτικών τελεστών σε αριθμητικές τιμές προς διευκόλυνση των υπολογισμών. Για παράδειγμα, έχουμε:

```
%%%%%%%%%%%%%%%%%DIMIOURGIA PINAKWN GIA TO PROVLIMA%%%%%%%%%%%%%%%%%%%%%%
%dimiourgia cell arrays vd kai vd2 gia ta zeugi variable-domains
%vd
vd=cell(400,2); %preallocation
for i=1:length(A.data)
    vd(i,1)={A.data(i,1)};
end
%topothetoume ston vd to antistoixo domain
for i=1:length(A.data)
    if A.data(i,2)==0
        vd(i,2)={B.data(1,2:end)};
    elseif A.data(i,2)==1
        vd(i,2)={B.data(2,2:29)};
    elseif A.data(i,2)==2
        vd(i,2)={B.data(3,2:24)};
    else
        vd(i,2)={B.data(4,2:8)};
    end
end
%vd2
vd2=cell(400,2); %preallocation vd2
vd2=repmat(vd,1);
% %ola 1 stin arxi ta domains ston vd2 / an ginei diagrafi vazw 0
for j=1:length(vd2)
    for k=1:length(vd2{j,2})
        vd2{j,2}(k)=1;
    end
end
fprintf('Creating Q...\n\n')
% antigrafw ton C.textdata ston C.textdata2
C.textdata2=repmat(C.textdata,1);
%kai ton antistrefw
C.textdata2(:,[1 2]) = C.textdata2(:,[2 1]);
%gia ton C.textdata
D={};
for i=1:length(C.textdata)
    D{i}={C.textdata{i,1:end}};
end
%gia ton C.textdata2
E={};
for i=1:length(C.textdata2)
    E{i}={C.textdata2{i,1:end}};
end
% dimiourgw tin Q ws enwsi twn epimerous cell arrays pou ftiaksame
Q=cell(8000,1); %preallocation memory for better speed
Q=cat(2,D,E)';
for i=1:length(Q)
    str2double(Q{i}(1));
```

Εικόνα6.3.2*~Δημιουργία δομής vd, μετατροπή τελεστών σε αριθμούς και δημιουργία ουράς Q*

Τρίτη και τελευταία ενότητα του κώδικα αποτελεί η καθ' εαυτή υλοποίηση του αλγορίθμου, μετά τις αρχικοποιήσεις διάφορων μετρητών και μεταβλητών. Όπως έχουμε αναλύσει σε προηγούμενα κεφάλαια, η εκτέλεση τόσο του AC-3 όσο και του RPC συνεχίζεται

όσο η ουρά Q δεν είναι άδεια, έχει δηλαδή τόξα να στείλει για εξέταση συνέπειας στις συναρτήσεις Revise και FindTwoSupports αντιστοίχως.

```
% %%%%%%%%%%%%%%%%%%%ARXH YLOPOIHSHS AC3%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function_revise_counter=0;
pruned_counter=0;
total_deletions_cntr=0;
re_enteringQ_cntr=0;
not_pruned_counter=0;
while not(isempty(Q)) %oso i oura Q den einai adeia
    for k=1:length(Q)
        x=Q(1,1); % to x   / tipos double
        y=Q(1,2); % to y   / tipos double
        operator=Q(1,3); %o telestis /tipos double
        stathera=Q(1,4); %kai i stathera /tipos double
        %stelnw ta antistoixa domains twn x kai y kathe fora
        for i=1:length(vd)
            if x==vd{i}(1)
                dom_x=vd{i,2};
                break;
            end
        end
        for i=1:length(vd)
            if y==vd{i}(1)
                dom_y=vd{i,2};
                break;
            end
        end
        %revise function
        [ delete_value,revised,NDx,NDy,x,y,deletions_cntr ] = revise(x,y,dom_x,dom_y,operator,stathera);
        function_revise_counter=function_revise_counter+1;
        Q(1,:)=[]; %diagrafw to prwto tokso tis ouras
        if revised==1
            pruned_counter=pruned_counter+1; %metritis gia diagrafes
            fprintf('\nArc (x%d,x%d) is not consistent, re-checking...\n',x,y);
            %ananewnw ta antistoixa domains twn x kai y kathe fora
            for i=1:length(vd)
                if x==vd{i}(1)
                    vd{i,2}=NDx;
                    break;
                end
            end
            for i=1:length(vd)
                if y==vd{i}(1)
                    vd{i,2}=NDy;
                    break;
                end
            end
```

Εικόνα6.3.3~*Αρχικοποίηση μετρητών, κλήση Revise συνάρτησης και διαγραφή τόξων από την Q*

6.3.1 Ανάλυση Υλοποίησης των Συναρτήσεων

Όπως πλέον ήδη γνωρίζουμε, οι αλγόριθμοι AC-3 και RPC, χρησιμοποιούν αυτές τις βασικές συναρτήσεις, ώστε να προβούν σε εξέταση των τόξων για συνέπεια τόξου ή μονοπατιού, αντίστοιχα, και αν δεν πληρούνται συγκεκριμένα κριτήρια, προχωρούν και σε διαγραφές τιμών από τα πεδία τιμών των εκάστοτε εξεταζόμενων τόξων.

6.3.1.1 Η συνάρτηση Revise

Την συνάρτηση αυτή την χρησιμοποιεί ο αλγόριθμος AC-3. Δέχεται ως ορίσματα εισόδου τις εκάστοτε (x_i, x_j) τιμές, τα αντίστοιχα domains τους για τυχόν πιθανές διαγραφές τιμών από αυτά, καθώς και τις τιμές του τελεστή και της σταθεράς, εφόσον υπάρχει.

```
14 -    for i=1:length(Dx)
15 -        for j=1:length(Dy)
16 -            if (~isnan(Dy(j)) && ~isnan(Dx(i)))
17 -                support_on_this_var=false;
18 -                if (abs(Dx(i)-Dy(j))==stathera)   %f10
19 -                    support_on_this_var=true;
20 -                    break;  %an vrei support break
21 -                end
22 -            end
23 -        end
24 -        if (~isnan(Dy(j)) && ~isnan(Dx(i)))
25 -            if support_on_this_var==false %diagrafi timis tis metavlitis x an den uparxei KANENA support stin y metavliti
26 -                Dx(i)=NaN; %timi i afaireitai apo to domain Dx
27 -                delete_value=true;
28 -                deletions_cntr=deletions_cntr+1;
29 -            end
30 -        end
31 -    end
32 -    revised=double(delete_value); %cast logical-->double
33 -    NDx=Dx; %ta nea domains twn metavlitwn pou epistrefontai apo ti revise sto kuriws programma
34 -    NDy=Dy; %ta nea domains twn metavlitwn pou epistrefontai apo ti revise sto kuriws programma
35 -    deletions_cntr=deletions_cntr;
```

Εικόνα6.3.1.1~*Η συνάρτηση Revise*

Εξετάζει σε ένα εμφωλευμένο for loop, προσπελάζοντας μία προς μία τις τιμές του domain της x_i, αρχίζοντας από την 1η, αν ικανοποιεί τον περιορισμό ή όχι με όλες τις τιμές από το domain της x_j. Αν βρει support, δηλαδή πληρείται ο περιορισμός, τότε κάνει break και προχωράμε στην επόμενη τιμή της x_i. Ειδάλλως, αν έχει ολοκληρώσει τους ελέγχους συνέπειας

για την 1η τιμή της x_i μεταβλητής με όλες τις τιμές της x_j μεταβλητής, και δεν έχουν ήδη διαγραφεί αυτές οι τρέχουσες τιμές από προηγούμενες κλήσεις της revise, τότε προβαίνει σε διαγραφή της τρέχουσας a_i τιμής από το πεδίο τιμών της $D(x_i)$. Αυτό υλοποιείται θέτοντας την τιμή NaN στην εν λόγω a_i , για λόγους διευκόλυνσης. Έπειτα, μετά το πέρας των ελέγχων και το τέλος και του εξωτερικού for, επιστρέφει την μεταβλητή revised (παίρνει τιμές 0 ή 1, 1 αν έγινε "revised" κάποιο από τα εξεταζόμενα τόξα, αλλιώς παίρνει την τιμή 0), καθώς και τα «νέα» πεδία τιμών των x_i και x_j μεταβλητών στον κύριο κώδικα του AC-3.

6.3.1.2 Η συνάρτηση FindTwoSupports

Παρόμοια λειτουργώντας ως την συνάρτηση Revise, η FindTwoSupports, που καλείται από τον κύριο κώδικα του RPC αλγορίθμου, έχει το ρόλο να εντοπίσει, εάν υπάρχει, το μοναδικό support για την τρέχουσα τιμή a_i της εκάστοτε x_i μεταβλητής. Για αυτό λοιπόν το λόγο χρησιμοποιούμε το flag support_for_PC. Στο εμφωλευμένο for loop εξετάζει μία προς μία τις τιμές του domain της x_i, αρχίζοντας από την 1η, αν ικανοποιεί τον περιορισμό ή όχι με όλες τις τιμές από το domain της x_j. Αν ικανοποιείται ο περιορισμός, το support_for_PC το θέτει ίσο με 1, έστω ότι βρήκε ένα support. Βέβαια, δε γνωρίζουμε ακόμη ότι είναι το μοναδικό support. Συνεχίζοντας την εκτέλεση, αν βρει πάλι support, θέτει το flag support_for_PC ίσο με 2 και κάνει break από το βρόγχο, σημάδι ότι βρήκε παραπάνω από ένα support για την εν λόγω a_i τιμή, και προχωράει στην επόμενη τιμή a_i.

Αν μετά το πέρας του εσωτερικού loop, έχει βρει μόνο ένα support, τότε και μόνο τότε θα καλέσει την pc_search συνάρτηση για να αναζητήσει pc_witness για το ζεύγος (a_i, a_j), δηλαδή να προχωρήσει σε έλεγχο συνέπειας μονοπατιού. Στην περίπτωση που δεν έχει βρεθεί τιμή support, δηλαδή το support_for_PC είναι ίσο με 0, τότε θα προχωρήσει σε διαγραφή της τρέχουσας a_i τιμής από το πεδίο τιμών της $D(x_i)$. Αν η pc_search επιστρέψει false, αυτό πρακτικά σημαίνει ότι η τιμή a_i δεν παρουσιάζει περιορισμένη συνέπεια μονοπατιού και πρέπει να διαγραφεί από το πεδίο τιμών της $D(x_i)$. Αλλιώς, αν επιστρέψει true, η τιμή a_i παρουσιάζει περιορισμένη συνέπεια μονοπατιού, οπότε γίνεται break και προχωράμε στην επόμενη τιμή a_i της μεταβλητής x_i. Ενδεικτικά, για το πρόβλημα f11, η συνάρτηση FindTwoSupports έχει ως εξής:

```
if (operator==0)
    for i=1:length(Dx)
        support_for_PC=0; %flag initialize
        for j=1:length(Dy)
            if (~isnan(Dy(j)) && ~isnan(Dx(i)))
                if (abs(Dx(i)-Dy(j))==stathera)   %f11
                    if support_for_PC == 0
                        support_for_PC=1; %vrethike ena support
                        one_support=Dy(j);   %thetoume stin one_support tin timi tou support
                    elseif support_for_PC == 1
                        support_for_PC=2;
                        break;
                    end
                end  %telos elegxou AC gia to zeugos (ai,aj)
            end
        end
        if (~isnan(Dx(i)) && ~isnan(Dy(j)))
            if support_for_PC == 1 %an vrethei monadiko support
                ai=Dx(i);
                aj=one_support;
                [pc_witness] = pc_search(x,y,var_con,vd,ai,aj,stathera,operator);
                if pc_witness==false
                    fprintf('\nValue ai=%d of x%d is not RPC.\n',ai,x);
                    not_rpc_cntr=not_rpc_cntr+1;
                    fprintf('Deleting value ai=%d from domain of x%d.\n',ai,x);
                    Dx(i)=NaN; %timi i afaireitai apo to domain Dx
                    delete_value=true;
                    deletions_cntr=deletions_cntr+1;
                else
                    fprintf('\nValue ai=%d of x%d is RPC.\n',ai,x);
                    rpc_cntr=rpc_cntr+1;
                    rpcwitness=double(pc_witness); %cast logical-->double
                    break;
                end
            elseif support_for_PC == 0
                Dx(i)=NaN; %timi i afaireitai apo to domain Dx giati de vrike kanena support
                delete_value=true; %ara den einai AC kai diagrafetai
                deletions_cntr=deletions_cntr+1;
            end
        end
    end
```

Εικόνα6.3.1.2~*Η συνάρτηση FindTwoSupports*

6.3.1.3 Η συνάρτηση pc_search

Η συνάρτηση pc_search αποτελεί αναπόσπαστο κομμάτι του RPC αλγορίθμου και είναι υπεύθυνη για την εύρεση pc_witness για το ζεύγος τιμών (a_i, a_j), εφόσον το a_j είναι και το μοναδικό support της a_i στο $D(x_j)$. Δέχεται ως ορίσματα τις τιμές του εξεταζόμενου τόξου (x_i, x_j), τις δομές var_con και vd και φυσικά τις τιμές a_i, a_j. Το μόνο όρισμα εξόδου το οποίο επιστρέφει στην FindTwoSupports είναι μία Boolean μεταβλητή pc_witness, η οποία λαμβάνει την τιμή true αν βρεθεί pc_witness- τουλάχιστον μία τιμή a_k για κάθε μεταβλητή x_k-, ενώ αν δε βρεθεί επιστρέφει false.

Αρχικά, σχηματίζει τη δομή Xκ που αναλύσαμε σε προηγούμενη παράγραφο, η οποία αποθηκεύει όλες τις κοινές μεταβλητές που συμμετέχουν από κοινού σε περιορισμό με τις τρέχουσες x_i, x_j μεταβλητές. Έπειτα, τα αντίστοιχα πεδία τιμών των κοινών μεταβλητών αποθηκεύονται στον cell array DXk. Η συνάρτηση λειτουργεί ως εξής:

```
    total_pc_witness_cntr=0;
    for ii=1:length(Xk)
        pc_witness_cntr=0; %metritis gia kathe metavliti xk
        for jj=1:length(DXk{1,ii})
            if (ai~=DXk{ii}(jj)) && (aj~=DXk{ii}(jj))  %elegxos AC gia ai,aj me oles tis times twn Xk metavlitwn
                pc_witness_cntr=pc_witness_cntr+1;  %gia tin kathe Xk
                total_pc_witness_cntr=total_pc_witness_cntr+1; %auksanetai o sinolikos metritis gia pc_witnesses
                break;  %ginetai break kai eksetazoume tin epomeni metavliti xk
            end
        end
        if pc_witness_cntr==0 %check the first xk variable
            break;
        end
    end
    if total_pc_witness_cntr>=length(Xk) %an uparxei toulaxiston mia i perissoteres pc_witness se kathe xk metavliti return true
        fprintf('\nPC-witness found for values ai=%d, aj=%d of arc (x%d,x%d)\n',ai,aj,x,y);
        pc_witness=true;
    else
        fprintf('\nNo pc-witness found for values ai=%d, aj=%d of arc (x%d,x%d)\n',ai,aj,x,y);
        pc_witness=false;
    end
```

Εικόνα6.3.1.3~*Η συνάρτηση pc_search*

Πριν το εσωτερικό for loop, δηλαδή για κάθε xk μεταβλητή, μηδενίζει έναν μετρητή ο οποίος αυξάνει κατά ένα, κάθε φορά που πληρούνται οι περιορισμοί συνέπειας, ταυτοχρόνως, για τις τιμές a_i, a_j και τις τιμές της εκάστοτε xk μεταβλητής. Αν πληρούνται, γίνεται break και εξετάζουμε την επόμενη μεταβλητή xk, διότι μας ενδιαφέρει η ύπαρξη έστω και μίας τιμής ως pc_witness για κάθε xk. Συνεπώς, εφόσον βρεθεί μία τέτοια μόνο, για λόγους ευχρηστίας και ταχύτητας, πάμε κατευθείαν στην επόμενη μεταβλητή xk.

Αν μετά το πέρας του εσωτερικού for loop, δεν έχει βρεθεί pc_witness, δηλαδή ο μετρητής παραμένει μηδέν, τότε κάνουμε break και αυτό σημαίνει ότι δεν υπάρχει pc_witness για το ζευγάρι τιμών a_i, a_j και επιστρέφουμε τη ροή εκτέλεσης στην FindTwoSupports.

Μετά το τέλος και του εξωτερικού loop, εξετάζουμε αν ο συνολικός μετρητής για pc_witness είναι τουλάχιστον ίσος ή μεγαλύτερος από το μήκος της δομής Xk, με άλλα λόγια αν υπάρχει τουλάχιστον μία ή περισσότερες τιμές a_k ως pc_witness σε κάθε xk μεταβλητή. Αν είναι όντως μεγαλύτερος ή ίσος, θέτουμε true την Boolean μεταβλητή και την επιστρέφουμε

στην FindTwoSupports, σε κάθε άλλη περίπτωση δεν έχει βρεθεί pc_witness, συνεπώς θέτουμε false την Boolean μεταβλητή και τερματίζει η συνάρτηση pc_search.

6.4 Εκτέλεση Προβλημάτων και Συγκεντρωτικά Αποτελέσματα

Η εκτέλεση των αλγορίθμων AC-3 και RPC πραγματοποιήθηκε σε υπολογιστή με επεξεργαστή AMD FX 8120 (4-core processor) χρονισμένο στα 3.1GHz, μνήμη ram 8GB DDR3 και 2 MB L2 cache.

Και με τους δύο αλγορίθμους, εκτελέσαμε πληθώρα προβλημάτων που ανήκουν στο χώρο των CSP και συγκεντρώσαμε τα αποτελέσματα τους, ώστε να μπορέσουμε να τα αναλύσουμε και να καταλήξουμε σε χρήσιμα και έγκυρα συμπεράσματα για την αποδοτικότητα και την ταχύτητα των αλγορίθμων AC-3 και RPC.

Βασικά χαρακτηριστικά που μας ενδιαφέρουν είναι το πλήθος διαγραφών τιμών από τα πεδία των μεταβλητών που πραγματοποιεί ο κάθε αλγόριθμος, ο χρόνος εκτέλεσης του αλγορίθμου σε δευτερόλεπτα, ο χρόνος επεξεργαστή (CPU time) και το πλήθος των τόξων που επανεξετάζονται από την Q. Η μετρική CPU time επιστρέφει τον συνολικό χρόνο του επεξεργαστή σε δευτερόλεπτα που χρησιμοποίησε η MATLAB από την αρχή εκτέλεσης του αλγορίθμου. Ο χρόνος εκτέλεσης σε δευτερόλεπτα διαφέρει από το cpu time και είναι χρησιμότερος για την εξαγωγή έγκυρων στατιστικών.

Ο πίνακας με τα συγκεντρωτικά αποτελέσματα από όλες τις εκτελέσεις αλγορίθμων, για ένα πλήθος 50 προβλημάτων ικανοποίησης δυαδικών περιορισμών, δίνεται παρακάτω. Να σημειωθεί ότι προβλήματα επισημασμένα με *, έχουν τροποποιηθεί με κατάλληλο τρόπο τα πεδία τιμών κάποιων τυχαίων μεταβλητών τους, ώστε να γίνονται διαγραφές τιμών από τους αλγορίθμους AC-3 και RPC.

	CSP	AC-3	RPC
homer-8*			
Total values deletions		25	38
Number of arcs re-entered in Q		336	359
CPU time(seconds)		$1.065487e^{+01}$	$1.209008e^{+01}$
Elapsed time(seconds)		4.731827	6.234871
f10			
Total values deletions		10740	6254
Number of arcs re-entered in Q		6323	1286
CPU time(seconds)		$1.874352e^{+02}$	$2.013661e^{+02}$
Elapsed time(seconds)		144.900001	189.313561
f11			
Total values deletions		2168	6151
Number of arcs re-entered in Q		1312	1286
CPU time(seconds)		$2.522536\ e^{+01}$	$2.250158\ e^{+02}$
Elapsed time(seconds)		16.056485	182.415854
school1-13 *			
Total values deletions		247	259
Number of arcs re-entered in Q		13200	13321
CPU time(seconds)		$5.241322\ e^{+01}$	$5.839273e^{+02}$
Elapsed time(seconds)		312.299940	349.584930
myciel6-7 *			
Total values deletions		34	43
Number of arcs re-entered in Q		305	387
CPU time(seconds)		$4.024826e^{+00}$	$4.383628e^{+00}$
Elapsed time(seconds)		2.017376	2.095479
fpsol2-i-2-25*			
Total values deletions		233	242
Number of arcs re-entered in Q		4847	4861
CPU time(seconds)		$1.340985e^{+02}$	$1.498390e^{+02}$
Elapsed time(seconds)		90.586777	95.843556
inithx-i-3-20*			
Total values deletions		144	151
Number of arcs re-entered in Q		1821	2006
CPU time(seconds)		$2.579477e^{+02}$	$2.982115e^{+02}$
Elapsed time(seconds)		149.733615	175.852197
inithx-i-1-15*			
Total values deletions		378	382
Number of arcs re-entered in Q		13164	13182
CPU time(seconds)		$5.767201e^{+02}$	$6.663583e^{+02}$

Elapsed time(seconds)	370.405741	437.325071
inithx-i-2-10*		
Total values deletions	123	123
Number of arcs re-entered in Q	2063	2063
CPU time(seconds)	$2.600069e^{+02}$	$2.784306e^{+02}$
Elapsed time(seconds)	145.240740	154.110301
mulsol-i-3-15*		
Total values deletions	95	95
Number of arcs re-entered in Q	748	748
CPU time(seconds)	$2.544376e^{+01}$	$2.769018e^{+01}$
Elapsed time(seconds)	19.071962	19.499410
zeroin-i-3-25*		
Total values deletions	219	251
Number of arcs re-entered in Q	748	772
CPU time(seconds)	$2.140334e^{+01}$	$2.591177e^{+01}$
Elapsed time(seconds)	15.810408	18.191584
zeroin-i-1-10*		
Total values deletions	143	164
Number of arcs re-entered in Q	2052	2114
CPU time(seconds)	$3.698784e^{+01}$	$4.110626e^{+01}$
Elapsed time(seconds)	25.397125	27.573348
zeroin-i-2-15*		
Total values deletions	157	173
Number of arcs re-entered in Q	966	984
CPU time(seconds)	$2.235494e^{+01}$	$2.542816e^{+01}$
Elapsed time(seconds)	16.237240	17.819087
myciel7-4_var*		
Total values deletions	101	116
Number of arcs re-entered in Q	1061	1104
CPU time(seconds)	$1.307288e^{+01}$	$1.555330e^{+01}$
Elapsed time(seconds)	8.482145	9.792232
huck-8*		
Total values deletions	27	29
Number of arcs re-entered in Q	116	125
CPU time(seconds)	$9.204059e^{-01}$	$1.107607e^{+00}$
Elapsed time(seconds)	0.537737	0.646339

anna-5*		
Total values deletions	27	32
Number of arcs re-entered in Q	188	202
CPU time(seconds)	$1.794012e^{+00}$	$2.028013e^{+00}$
Elapsed time(seconds)	1.011175	1.141504
lei450-25a-22*		
Total values deletions	253	279
Number of arcs re-entered in Q	2400	2515
CPU time(seconds)	$7.377287e^{+01}$	$9.194699e^{+01}$
Elapsed time(seconds)	46.667970	55.980248
lei450-25d-10*		
Total values deletions	399	421
Number of arcs re-entered in Q	14590	15325
CPU time(seconds)	$6.763423e^{+02}$	$6.845948e^{+02}$
Elapsed time(seconds)	379.781483	381.209161
lei450-25c-10*		
Total values deletions	261	293
Number of arcs re-entered in Q	10790	9479
CPU time(seconds)	$4.811851e^{+02}$	$4.489709e^{+02}$
Elapsed time(seconds)	266.885529	245.817235
2-fullins-4-4_var*		
Total values deletions	84	125
Number of arcs re-entered in Q	545	942
CPU time(seconds)	$7.129246e^{+00}$	$1.148167e^{+01}$
Elapsed time(seconds)	4.216775	6.578997
5-fullins-4-6_var*		
Total values deletions	83	146
Number of arcs re-entered in Q	646	1378
CPU time(seconds)	$8.408454e^{+01}$	$1.378581e^{+02}$
Elapsed time(seconds)	46.156508	76.373252
4-insertions-4-3*		
Total values deletions	45	78
Number of arcs re-entered in Q	129	505
CPU time(seconds)	$6.817244e^{+00}$	$1.786211e^{+01}$
Elapsed time(seconds)	3.636138	9.416003
3-insertions-5-3*		
Total values deletions	78	136
Number of arcs re-entered in Q	481	777
CPU time(seconds)	$7.834370e^{+01}$	$1.879968e^{+02}$

Elapsed time(seconds)	42.717434	119.158346
games120-9*		
Total values deletions	52	71
Number of arcs re-entered in Q	220	264
CPU time(seconds)	$2.355615e^{+00}$	$3.135620e^{+00}$
Elapsed time(seconds)	1.260197	1.712904
miles1000-25*		
Total values deletions	185	230
Number of arcs re-entered in Q	2112	2052
CPU time(seconds)	$2.500696e^{+01}$	$2.875098e^{+01}$
Elapsed time(seconds)	16.090079	17.989788
miles500-5*		
Total values deletions	73	80
Number of arcs re-entered in Q	720	707
CPU time(seconds)	$5.803237e^{+00}$	$6.676843e^{+00}$
Elapsed time(seconds)	3.335532	3.774058
queen16-16-17*		
Total values deletions	179	283
Number of arcs re-entered in Q	3190	3360
CPU time(seconds)	$6.319601e^{+01}$	$7.974771e^{+01}$
Elapsed time(seconds)	42.332403	53.464395
queen10-10-12*		
Total values deletions	86	98
Number of arcs re-entered in Q	1031	1277
CPU time(seconds)	$8.377254e^{+00}$	$1.127887e^{+01}$
Elapsed time(seconds)	4.998690	6.499589
miles750-15*		
Total values deletions	118	154
Number of arcs re-entered in Q	1654	1140
CPU time(seconds)	$1.834572e^{+01}$	$1.450809e^{+01}$
Elapsed time(seconds)	10.814743	8.362817
school1-nsh-13*		
Total values deletions	124	305
Number of arcs re-entered in Q	2789	10842
CPU time(seconds)	$1.348317e^{+02}$	$3.869449e^{+02}$
Elapsed time(seconds)	79.759385	223.915166

school1-14*		
Total values deletions	142	332
Number of arcs re-entered in Q	5561	14972
CPU time(seconds)	$2.669645e^{+02}$	$7.103974e^{+02}$
Elapsed time(seconds)	144.628414	407.962012
mug88-25-4*		
Total values deletions	14	17
Number of arcs re-entered in Q	20	25
CPU time(seconds)	$5.304034e^{-01}$	$5.772037e^{-01}$
Elapsed time(seconds)	0.292748	0.342441
mug100-25-3*		
Total values deletions	12	15
Number of arcs re-entered in Q	18	25
CPU time(seconds)	$4.524029e^{-01}$	$1.076407e^{+00}$
Elapsed time(seconds)	0.301439	0. 583668
ash958GPIA-3*		
Total values deletions	31	64
Number of arcs re-entered in Q	110	660
CPU time(seconds)	$1.259708e^{+02}$	$2.279955e^{+02}$
Elapsed time(seconds)	64.260302	131.250444
ash608GPIA-3*		
Total values deletions	31	64
Number of arcs re-entered in Q	110	334
CPU time(seconds)	$5.357074e^{+01}$	$1.046923e^{+02}$
Elapsed time(seconds)	28.062819	61.875164
3-fullins-5-5*		
Total values deletions	96	189
Number of arcs re-entered in Q	1291	2626
CPU time(seconds)	$4.869571e^{+02}$	$7.543584e^{+02}$
Elapsed time(seconds)	260.547741	414.765439
4-fullins-3-6*		
Total values deletions	32	37
Number of arcs re-entered in Q	171	209
CPU time(seconds)	$1.856412e^{+00}$	$2.511616e^{+00}$
Elapsed time(seconds)	1.023982	1.346729
lei450-05d-05*		
Total values deletions	114	171
Number of arcs re-entered in Q	2135	3129
CPU time(seconds)	$7.394447e^{+01}$	$1.138651e^{+02}$

Elapsed time(seconds)	46.218520	69.037109
lei450-15c-08*		
Total values deletions	187	287
Number of arcs re-entered in Q	446	10741
CPU time(seconds)	$3.131252e^{+02}$	$4.661778e^{+02}$
Elapsed time(seconds)	172.351159	267.558394
myciel7-8*		
Total values deletions	82	103
Number of arcs re-entered in Q	1124	1217
CPU time(seconds)	$1.279208e^{+01}$	$1.680131e^{+01}$
Elapsed time(seconds)	7.782524	10.226036
fpsol2-i-1-20*		
Total values deletions	286	231
Number of arcs re-entered in Q	11641	4874
CPU time(seconds)	$2.921119e^{+02}$	$2.511304e^{+02}$
Elapsed time(seconds)	188.386270	147.424909
mulsol-i-2-25*		
Total values deletions	219	232
Number of arcs re-entered in Q	3858	1776
CPU time(seconds)	$4.488149e^{+01}$	$3.063860e^{+01}$
Elapsed time(seconds)	28.639038	21.016868
miles250-8*		
Total values deletions	39	47
Number of arcs re-entered in Q	158	178
CPU time(seconds)	$1.372809e^{+00}$	$1.840812e^{+00}$
Elapsed time(seconds)	0.782253	1.002593
homer-10*		
Total values deletions	1	40
Number of arcs re-entered in Q	561	359
CPU time(seconds)	$2.792418e^{+00}$	$1.168447e^{+01}$
Elapsed time(seconds)	1.479561	6.093591
david-8*		
Total values deletions	22	22
Number of arcs re-entered in Q	209	209
CPU time(seconds)	$1.404009e^{+00}$	$1.731611e^{+00}$
Elapsed time(seconds)	0.788122	0.964553

inithx-i-3-15*		
Total values deletions	263	293
Number of arcs re-entered in Q	3882	4317
CPU time(seconds)	$2.636261e^{+02}$	$2.977903e^{+02}$
Elapsed time(seconds)	158.376993	183.993812
inithx-i-2-05*		
Total values deletions	92	111
Number of arcs re-entered in Q	1191	1607
CPU time(seconds)	$2.452336e^{+02}$	$2.737350e^{+02}$
Elapsed time(seconds)	144.191827	161.020958
zeroin-i-3-10*		
Total values deletions	103	114
Number of arcs re-entered in Q	696	748
CPU time(seconds)	$2.138774e^{+01}$	$2.424256e^{+01}$
Elapsed time(seconds)	16.082639	17.520540
queen14-14-12*		
Total values deletions	202	253
Number of arcs re-entered in Q	3020	3471
CPU time(seconds)	$4.319668e^{+01}$	$5.923358e^{+01}$
Elapsed time(seconds)	27.923203	37.035452
queen15-15-16*		
Total values deletions	244	298
Number of arcs re-entered in Q	2818	3902
CPU time(seconds)	$5.093433e^{+01}$	$7.279007e^{+01}$
Elapsed time(seconds)	33.863110	47.612961
ash331GPIA-3*		
Total values deletions	70	101
Number of arcs re-entered in Q	421	1217
CPU time(seconds)	$2.199614e^{+01}$	$6.686203e^{+01}$
Elapsed time(seconds)	12.309701	39.557706

Ο συγκεντρωτικός πίνακας που περιέχει τη μέση τιμή των διαγραφών τιμών που πραγματοποίησαν οι αλγόριθμοι κατά την εκτέλεση τους, όπως επίσης και η μέση τιμή του χρόνου εκτέλεσης τους, δίνεται παρακάτω:

	AC-3	RPC
Mean values deletions	382.64	403.18
Mean elapsed time(seconds)	71	98

Αυτές οι δύο τιμές μας ενδιαφέρουν περισσότερο. Ο AC-3 πραγματοποίησε, κατά μέσο όρο, στα 50 προβλήματα που εκτελέσαμε 382.64 διαγραφές τιμών, ενώ ο RPC 403.18 διαγραφές αντίστοιχα. Όσον αφορά τον χρόνο εκτέλεσης τους, ο AC-3 διήρκησε, κατά μέσο όρο, 71 δευτερόλεπτα, ενώ ο RPC διήρκησε 98 δευτερόλεπτα.

Κεφάλαιο 7

7.1 Συμπεράσματα

Φτάνοντας στο 7^{ο} και τελευταίο κεφάλαιο της διπλωματικής εργασίας, έχοντας αναλύσει, μελετήσει σε θεωρητικό επίπεδο και έχοντας υλοποιήσει σε πρακτικό επίπεδο τους αλγορίθμους AC-3 και RPC, μπορούμε, πλέον, να καταλήξουμε σε ορισμένα συμπεράσματα που προκύπτουν αναφορικά με την αποτελεσματικότητα και την αποδοτικότητα των αλγορίθμων αυτών.

Αν εξετάσουμε τους συγκεντρωτικούς πίνακες αποτελεσμάτων από τις εκτελέσεις των αλγορίθμων στα εισαγόμενα CSP προβλήματα, γίνεται αντιληπτό πως ο RPC υπερέχει αισθητά του AC-3 όσον αφορά τις διαγραφές τιμών από τα πεδία μεταβλητών. Με άλλα λόγια, ο RPC επιτυγχάνει καλύτερο pruning τιμών, συγκριτικά με τον AC-3, συνεπώς, σε πραγματικά προβλήματα ικανοποίησης περιορισμών, θα ήταν προτιμότερος για να περιορίσει τον χώρο αναζήτησης. Σαφώς, ο ευρέως χρησιμοποιούμενος AC-3 δεν είναι σε καμία των περιπτώσεων «μη-υπολογίσιμη δύναμη», όσον αφορά τον περιορισμό του χώρου αναζήτησης. Ωστόσο, ο RPC, υλοποιημένος κατά αυτόν τον τρόπο, ώστε να πετυχαίνει περιορισμένη συνέπεια μονοπατιού, αποτελεί καλύτερη επιλογή συγκριτικά με τον AC-3.

Από την άποψη της απόδοσης του χρόνου εκτέλεσης, ο AC-3 έρχεται πρώτος και αφήνει με αξιοσημείωτη διαφορά τον RPC στη δεύτερη θέση. Όπως φαίνεται και στον συγκεντρωτικό πίνακα στην προηγούμενη ενότητα, ο μέσος χρόνος εκτέλεσης των προβλημάτων από τον AC-3 είναι 71 δευτερόλεπτα, εν αντιθέσει με τον RPC ο οποίος κατά μέσο όρο διήρκησε 98 δευτερόλεπτα. Λόγου χάρη, σε προβλήματα πιο πολύπλοκα, δηλαδή με πολλούς περιορισμούς όπως το ash331GPIA-3 ή το school1-14, η διαφορά φαίνεται αισθητά. Σε προβλήματα με λιγότερους περιορισμούς, όπως είναι τα homer προβλήματα, η διαφορά στο χρόνο εκτέλεσης δεν είναι τόσο αισθητή. Αυτή η χειρότερη χρονική απόδοση του RPC μπορεί εν μέρει να δικαιολογηθεί αφενός από το γεγονός ότι οι δομές δεδομένων που χρησιμοποιεί κατά την εκτέλεση του είναι πιο πολύπλοκες συγκριτικά με αυτές του AC-3 και αφετέρου από την ίδια του την δομή ως αλγόριθμος. Ο RPC επιτυγχάνει περιορισμένη συνέπεια μονοπατιού και συνεπώς χρειάζεται πιο πολύ χρόνο ώστε να πραγματοποιήσει τους υπολογισμούς για ένα προς

ένα τα ζεύγη (a_i, a_j) για τα οποία βρίσκει, εφόσον είναι arc-consistent πρώτα, πως η τιμή a_j είναι το μοναδικό support της a_i στο $D(x_j)$.

Εν κατακλείδι, αν είναι να επιλέξουμε μεταξύ των δύο αλγορίθμων, AC-3 και RPC, για να τον εφαρμόσουμε σε κάποιο πρόβλημα ικανοποίησης δυαδικών περιορισμών, θα πρέπει να αναλογιστούμε τι προτεραιότητες έχουν τεθεί. Αν κριτήριο αποτελεί η ταχύτερη, αλλά όχι βέλτιστη επίλυση του προβλήματος, τότε θα επιλεγεί ο αλγόριθμος AC-3. Αν, όμως, το βασικό κριτήριο είναι η βέλτιστη απόδοση και ο - κατά το μέγιστο δυνατό - περιορισμός του χώρου αναζήτησης, τότε η επιλογή δεν είναι άλλη από τον αλγόριθμο RPC.

Βιβλιογραφία

[1] “Artificial Intelligence: A Modern Approach Second Edition”, Stuart J.Russel and Peter Norvig, Chapter 5

[2] A.K. Mackworth. Consistency in networks of relations. Technical Report 75-3, Dept. of Computer Science, Univ. of B.C. Vancouver, 1975

[3] “Restricted Path Consistency Revisited”, Kostas Stergiou, Department of Informatics and Telecommunications Engineering, University of Western Macedonia, Greece

[4] “From Restricted Path Consistency to Max Restricted Path Consistency”, Romuald Debruyne and Christian Bessiere

[5] «Τεχνητή Νοημοσύνη», Κεφάλαιο 6^{ο}, των Βλαχάβα, Κεφάλα, Βασιλειάδη, Κόκκορα και Σακελλαρίου

[6] R. Mohr and T.C. Henderson. “Arc and path consistency revisited”. Artificial Intelligence, 28:225{233, 1986}

[7] Y. Zhang and R.H.C. Yap. “Making AC-3 an optimal algorithm” In Proceedings IJCAI'01, pages 316{321, Seattle WA, 2001.