ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

"Τοπική Ακτινωτή Αναζήτηση για Προβλήματα Ικανοποίησης Περιορισμών"

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ
ΤΗΣ

ΚΑΛΟΓΙΑΝΝΙΔΟΥ ΕΥΜΟΡΦΙΑΣ

ΑΕΜ:67

Επιβλέπων Καθηγητής: Κ. Στεργίου

ΠΕΡΙΛΗΨΗ

Σκοπός της διπλωματικής εργασίας είναι η εξέταση της απόδοσης του αλγορίθμου Τοπικής Ακτινικής Αναζήτησης(Local Beam search), ένος ευριστικού αλγορίθμου τοπικής αναζήτησης, για την εύρεση λύσεων σε προβλήματα ικανοποίησης περιορισμών. Επίσης, υλοποιήσαμε τον αλγόριθμο τοπικής αναζήτησης min-conflicts(Ελάχιστων Συγκρούσεων) με στόχο την σύγκριση των δύο αλγορίθμων.

Συγκεκριμένα, οι δύο αλγόριθμοι αναλύθηκαν και στη συνέχεια υλοποιήθηκαν, στη γλώσσα προγραμματισμού C++, και εκτελέστηκαν επανειλημμένα για γνωστά προβλήματα ικανοποίησης περιορισμών, διαφόρων μεγεθών. Συγκεντρώνοντας και εξετάζοντας τα αποτελέσματα, καταλήγουμε σε συμπεράσματα για την απόδοση των δύο αλγορίθμων.

Ουσιαστικά η διπλωματική αυτή απαντά στην ερώτηση αν ο αλγόριθμος Τοπικής Ακτινικής Αναζήτησης, ο οποίος είναι μια εξελιγμένη μορφή του αλγορίθμου Ελάχιστων Συγκρούσεων, αποτελεί αποδοτικότερο αλγόριθμο, όχι μόνο στη θεωρία αλλά και στη πράξη. Κάτι το οποίο αποδεικνύουμε ότι ισχύει.

ΠΕΡΙΕΧΟΜΕΝΑ

"Τοπική Ακτινωτή Αναζήτηση για Προβλήματα Ικανοποίησης Περιορισμών"....1
1.ΕΙΣΑΓΩΓΗ....5
1.1 Επίλυση Προβλημάτων – Τεχνητή Νοημοσύνη....5
1.2 Αντικείμενο Διπλωματικής....11
2. ΘΕΩΡΗΤΙΚΟ ΥΠΟΒΑΘΡΟ....12
2.1 Τοπική Αναζήτηση....12
2.2 Προβλήματα Ικανοποίησης Περιορισμών....17
2.3 Τοπική Αναζήτηση στα Προβλήματα περιορισμών....21
3.ΥΛΟΠΟΙΗΣΗ ΑΛΓΟΡΙΘΜΩΝ....25
3.1 Κύριο πρόγραμμα....25
3.2 Αλγόριθμος Ελάχιστων Συγκρούσεων....26
3.3 Αλγόριθμος Ακτινικής Τοπικής Αναζήτησης....29
4.ΑΠΟΤΕΛΕΣΜΑΤΑ....33
4.1.Αποτελέσματα αλγορίθμου Ελάχιστων Συγκρούσεων....33
4.2.Αποτελέσματα αλγορίθμου Ακτινικής Αναζήτησης....36
5.ΣΥΜΠΕΡΑΣΜΑΤΑ....39

1.ΕΙΣΑΓΩΓΗ

1.1 Επίλυση Προβλημάτων – Τεχνητή Νοημοσύνη

Ο όρος Τεχνητή Νοημοσύνη αναφέρεται στον κλάδο της επιστήμης υπολογιστών ο οποίος ασχολείται με τη σχεδίαση και την υλοποίηση υπολογιστικών συστημάτων που μιμούνται στοιχεία της ανθρώπινης συμπεριφοράς τα οποία υπονοούν έστω και στοιχειώδη ευφυΐα: μάθηση, προσαρμοστικότητα, εξαγωγή συμπερασμάτων, κατανόηση από συμφραζόμενα και επίλυση προβλημάτων.

Η επίλυση προβλημάτων είναι τμήμα της διαδικασίας της σκέψης. Θεωρείται η πλέον περίπλοκη διανοητική λειτουργία και έχει οριστεί ως υψηλού επιπέδου γνωστική διεργασία, η οποία απαιτεί τον χειρισμό και έλεγχο περισσότερο θεμελιωδών και συνηθισμένων ικανοτήτων. Λαμβάνει χώρα όταν ένας βιολογικός οργανισμός ή σύστημα τεχνητής νοημοσύνης δεν γνωρίζει πώς να μεταβεί από μία δεδομένη κατάσταση σε μία επιθυμητή κατάσταση στόχο. Αποτελεί κλάδο της τεχνητής νοημοσύνης (ΤΝ) ο οποίος αφορά τον σχεδιασμό κατάλληλων ενεργειών με στόχο την άφιξη ενός ελέγξιμου συστήματος σε μία αποδεκτή τελική κατάσταση, εκκινώντας από κάποια προκαθορισμένη αρχική κατάσταση. Συνήθως αυτό γίνεται μέσω ενός αλγορίθμου ο οποίος λαμβάνει ως είσοδο το δοθέν πρόβλημα και επιστρέφει ως έξοδο μία λύση σε αυτό, αφού αξιολογήσει πρώτα μία ομάδα υποψηφίων λύσεων.

Ένα πρόβλημα είναι ένα σύνολο αντικειμένων, ιδιοτήτων και σχέσεων το οποίο ορίζεται από μία αρχική κατάσταση, μία επιθυμητή τελική κατάσταση και τις επιτρεπτές ενέργειες στα αντικείμενα του προβλήματος. Στόχος είναι, ξεκινώντας από την αρχική κατάσταση, να γίνει μία κατάλληλη ακολουθία ενεργειών η οποία να καταλήγει στην τελική κατάσταση. Αυτή η διαδικασία ονομάζεται επίλυση του προβλήματος Βασικό στοιχείο στην επίλυση προβλημάτων είναι η αναπαράσταση τους και γι' αυτό το σκοπό υπάρχουν δύο μεθοδολογίες: η αναπαράσταση με χώρο καταστάσεων και η αναπαράσταση με αναγωγή. Στη μέθοδο της αναγωγής δομική μονάδα περιγραφής του προβλήματος είναι η ίδια η περιγραφή αναλυόμενη σε πολλαπλές, απλούστερες εκδοχές. Αυτή η ανάλυση συμβαίνει διαδοχικά ώσπου να καταλήξει σε αρχέγονα προβλήματα επιλυόμενα με προφανή τρόπο. Προγραμματιστικά η αναγωγή υλοποιείται με αναδρομή και κεντρική έννοια σε αυτήν αποτελούν οι τελεστές αναγωγής, διαδικασίες οι οποίες ανάγουν ένα πρόβλημα σε υποπροβλήματα.

Στη μέθοδο χώρου καταστάσεων βασική δομική μονάδα είναι η κατάσταση, το σύνολο δηλαδή των αντικειμένων που εμπλέκονται στο πρόβλημα συν τις ιδιότητες τους και τις μεταξύ τους σχέσεις. Η κατάσταση ορίζεται σε ένα απλουστευμένο, αφαιρετικό μοντέλο του κόσμου και το σύνολο των καταστάσεων (στιγμιότυπων) στις οποίες μπορεί να βρεθεί αυτός ο κόσμος του προβλήματος ονομάζεται χώρος καταστάσεων. Το ίδιο το πρόβλημα ορίζεται με βάση την αρχική κατάσταση από την οποία ξεκινάμε,

την επιθυμητή τελική κατάσταση στην οποία πρέπει να καταλήξουμε (ή πολλές δυνατές τελικές) και το σύνολο των τελεστών μετάβασης, δηλαδή επιτρεπτών πράξεων που μπορούν να εκτελεστούν στα αντικείμενα μίας κατάστασης οδηγώντας σε μια άλλη. Λύση του προβλήματος είναι μία ακολουθία διαδοχικών τελεστών μετάβασης και καταστάσεων που ξεκινά από μία αρχική κατάσταση και καταλήγει σε μία τελική.

Υπάρχει μία πλειάδα πραγματικών ή συνθετικών, απλών ή πολύπλοκων προβλημάτων που μπορούν να αναπαρασταθούν με χώρο καταστάσεων. Όλα τα προηγούμενα βρίσκουν εφαρμογή και το μόνο που αλλάζει σε κάθε πρόβλημα είναι οι λεπτομέρειες (οι ιδιότητες των αντικειμένων, οι επιτρεπτοί τελεστές κλπ). Προκειμένου ένα πρόγραμμα να επιλύσει ένα τέτοιο πρόβλημα πρέπει να αναπαραστήσει κατάλληλα και να κατασκευάσει το δένδρο των καταστάσεων, ξεκινώντας από τη ρίζα και επεκτείνοντας τους κόμβους μέχρι να φτάσει σε κάποια τελική κατάσταση. Αν το ζητούμενο είναι να βρεθεί μία οποιαδήποτε λύση τότε το πρόγραμμα μπορεί τότε να τερματίσει επιστρέφοντας το μονοπάτι που οδηγεί στο τρέχον φύλλο, διαφορετικά (εξαντλητική αναζήτηση) μπορεί να αποθηκεύσει έναν δείκτη προς αυτό το φύλλο και να συνεχίσει την κατασκευή του δένδρου μέχρι να ανακαλύψει όλες τις πιθανές καταστάσεις που είναι προσβάσιμες από την αρχική, με τους διαθέσιμους τελεστές μετάβασης, και όλες τις πιθανές λύσεις.

Υπάρχει ένας γενικός αλγόριθμος αναζήτησης που εκτελεί αυτήν τη διερεύνηση και οι πραγματικοί αλγόριθμοι που χρησιμοποιούνται είναι παραλλαγές του. Στον αλγόριθμο αυτό Μέτωπο Αναζήτησης (Μ.Α.) είναι το σύνολο των καταστάσεων που έχουμε επισκεφθεί αλλά δεν έχουμε επεκτείνει και Κλειστό Σύνολο (Κ.Σ.) το σύνολο των καταστάσεων που και έχουμε επισκεφθεί και έχουμε επεκτείνει. Το Κ.Σ. είναι απαραίτητο μόνο αν υπάρχει κίνδυνος παγίδευσης του αλγορίθμου σε ατέρμονα βρόχο λόγω απείρου μήκους κλαδιών στο δένδρο.

Οι διάφορες πραγματικές παραλλαγές αυτού του αλγορίθμου διακρίνονται σε αλγορίθμους τυφλής αναζήτησης, που διατάσσουν το Μ.Α. αποκλειστικά με βάση το χρόνο δημιουργίας κάθε κόμβου κατά την κατασκευή του δένδρου, και σε αλγορίθμους ευρετικής αναζήτησης (heuristic search), όπου εξαρτώνται από μία επιπλέον πληροφορία που υπολογίζεται σε πραγματικό χρόνο και που στις περισσότερες περιπτώσεις, αλλά όχι πάντα, είναι σχετικά ακριβής και αξιολογεί προσεγγιστικά τις καταστάσεις σε «καλές» και «κακές». Ένα παράδειγμα ευρετικής πληροφορίας που μπορεί να αντιστοιχιστεί σε κάθε ενδιάμεση κατάσταση είναι η εκτιμώμενη «απόστασή» της (με βάση ένα μέτρο που εξαρτάται από το πρόβλημα και την υλοποίηση) από την τελική. Έτσι μπορούμε, φερ' ειπείν, να κλαδεύουμε τα υποδένδρα με ρίζα «κακή» κατάσταση, αφαιρώντας τη ρίζα τους από το Μ.Α. προτού την επεκτείνουμε. Προφανώς αυτή η τακτική συμβάλλει στην αντιμετώπιση του φαινομένου της συνδυαστικής έκρηξης.

Μία άλλη κατηγοριοποίηση των αλγορίθμων γίνεται ανάλογα με τον τύπο του προβλήματος που επιλύουν: εκτός από τα συνηθισμένα που προαναφέρθηκαν, υπάρχουν και προβλήματα βελτιστοποίησης (όπου σε

κάθε τελεστή μετάβασης αντιστοιχίζεται μία τιμή κόστους και αναζητούμε τη λύση με το μονοπάτι που αθροιστικά έχει το ελάχιστο κόστος) ή προβλήματα ικανοποίησης περιορισμών (όπου η τελική κατάσταση δεν είναι πλήρως γνωστή, γνωρίζουμε όμως κάποιες ιδιότητες της και επιθυμούμε να καταλήξουμε σε μία κατάσταση που να τις διαθέτει). Πληρότητα ενός αλγορίθμου αναζήτησης ονομάζεται το κατά πόσον βρίσκει πάντα μία λύση, εφ' όσον τέτοια υπάρχει. Πιο συγκεκριμένα:

Τυφλή αναζήτηση

Οι σπουδαιότεροι αλγόριθμοι τυφλής αναζήτησης είναι ο DFS (Depth-First Search ή αναζήτηση κατά βάθος) και ο BFS (Breadth-First Search ή αναζήτηση κατά πλάτος), οι οποίοι κατασκευάζουν το δένδρο ξεκινώντας από τη ρίζα και παράγοντας κόμβους, ο μεν DFS κατά βάθος (ακολουθεί ένα κλαδί μέχρι να φτάσει σε φύλλο και μετά επεκτείνει έναν κόμβο προηγούμενου επιπέδου· αυτή η μέθοδος ονομάζεται «οπισθοδρόμηση»), ο δε BFS κατά πλάτος (επεκτείνει πρώτα όλους τους κόμβους ενός επιπέδου, οι οποίοι έχουν το ίδιο βάθος, και μετά προχωρά στους κόμβους του επόμενου επιπέδου). Προγραμματιστικά είναι σχεδόν ίδιοι μεταξύ τους, αλλά και με το γενικό αλγόριθμο που περιγράφηκε προηγουμένως, μόνο που διαφέρουν στο βήμα 8 (το βήμα 7 δεν υπάρχει αφού δε γίνεται κλάδεμα): ο DFS τοποθετεί τους νέους κόμβους που προστίθενται στο Μ.Α. στην αρχή της λίστας (LIFO στοίβα), ώστε στην επόμενη επανάληψη του βρόχου να επεκταθεί ένας από αυτούς, ενώ ο BFS τους τοποθετεί στο τέλος της λίστας (FIFO ουρά), ώστε στην επόμενη επανάληψη του βρόχου να επεκταθεί ένας «αδελφός» του γονέα τους αν υπάρχει.

Ο BFS εγγυάται ότι θα βρει πρώτα τη λύση με την ελάχιστη απόσταση από τη ρίζα (οπότε είναι ιδανικός και για προβλήματα βελτιστοποίησης όπου όλοι οι τελεστές έχουν ίσο κόστος) και είναι πλήρης, το Μ.Α. όμως μπορεί να γιγαντωθεί για μεγάλους χώρους αναζήτησης και άρα έχει μεγάλες απαιτήσεις σε μνήμη. Από την άλλη ο DFS είναι τυχαίο το ποια λύση θα βρει πρώτα και δεν είναι πλήρης, καθώς αν δε χρησιμοποιείται Κλειστό Σύνολο μπορεί να παγιδευτεί σε κλαδιά απείρου μήκους (αφού ακολουθεί ένα κλαδί μέχρι να καταλήξει σε φύλλο). Από την άλλη έχει μικρές απαιτήσεις σε μνήμη διατηρώντας πάντα μικρό το Μ.Α.

Συμβιβασμό μεταξύ αυτών των δύο αποτελεί ο αλγόριθμος ID (Iterative Deepening ή επαναληπτική εκβάθυνση), ο οποίος είναι κατά βάση DFS αλλά προχωρά μέχρι ένα προκαθορισμένο βάθος, ενώ στη συνέχεια το επιτρεπτό βάθος αυξάνεται και ο αλγόριθμος ξεκινά από την αρχή χωρίς να διατηρεί δεδομένα από την προηγούμενη αναζήτηση. Το δένδρο δηλαδή κατασκευάζεται διαρκώς από τη ρίζα, ξανά και ξανά, αλλά σε όλο και μεγαλύτερο βάθος. Παρ' όλο που ο ID εκτελεί πολλή περιττή εργασία αυτό δεν παίζει ρόλο σε μεγάλους χώρους αναζήτησης όσον αφορά την αλγοριθμική πολυπλοκότητα. Ο αλγόριθμος είναι πλήρης γιατί δεν μπορεί να παγιδευτεί σε άπειρα κλαδιά, αφού το βάθος αναζήτησης είναι προκαθορισμένο, έχει τις μικρές απαιτήσεις μνήμης του DFS, ενώ αν το επιτρεπτό βάθος σε κάθε επανάληψη αυξάνεται κατά 1 εγγυάται ότι θα βρει

πρώτα τη λύση με την ελάχιστη απόσταση από τη ρίζα (όπως ο BFS, αφού αν υπήρχε καλύτερη λύση θα βρισκόταν σε προηγούμενη επανάληψη).

Οποιοσδήποτε από αυτούς τους αλγορίθμους μπορεί να χρησιμοποιηθεί με τη μέθοδο BiS (Bidirectional Search ή αμφίδρομη αναζήτηση), η οποία μπορεί να εφαρμοστεί σε υπολογιστικό σύστημα με δύο επεξεργαστές όταν η τελική κατάσταση είναι πλήρως γνωστή και οι τελεστές μετάβασης είναι αντιστρέψιμοι: ο ένας επεξεργαστής εκτελεί αναζήτηση από την αρχική προς την τελική κατάσταση και ο άλλος από την τελική προς την αρχική. Όταν βρεθεί μία κοινή κατάσταση το πρόγραμμα ενώνει τα δύο μονοπάτια και επιστρέφει την τελική λύση· ιδανικά στο 1/2 του χρόνου που θα απαιτούσε μία μονόδρομη αναζήτηση.

Σε προβλήματα βελτιστοποίησης με τελεστές διαφορετικού (αλλά πάντα θετικού) κόστους μπορεί να εφαρμοστεί ο αλγόριθμος τυφλής αναζήτησης B&B (Branch and Bound ή επέκταση και οριοθέτηση), ο οποίος μπορεί να βασιστεί είτε στον DFS είτε στον BFS προσφέροντας όμως επιπλέον κλάδεμα των καταστάσεων -και των αντίστοιχων υποδένδρων που θα προέκυπταν από την επέκταση τους- που αποκλείεται να οδηγούν σε λύση καλύτερη από την τρέχουσα. Για να το πετύχει αυτό κρατά σε μία μεταβλητή Β το ολικό κόστος του μονοπατιού της βέλτιστης λύσης που έχει βρεθεί ως τώρα και, αν το μονοπάτι του τρέχοντος ενδιάμεσου κόμβου έχει κόστος μεγαλύτερο του Β, δεν τον αναπτύσσει και τον αφαιρεί από το Μέτωπο Αναζήτησης. Στη χειρότερη περίπτωση δε θα γίνει κανένα κλάδεμα, αφού είναι θέμα τύχης η σειρά με την οποία θα ανακαλυφθούν οι λύσεις, και ο B&B λειτουργεί όπως ο DFS ή ο BFS.

Ευρετική αναζήτηση

Προκειμένου να μειωθεί ο, γιγάντιος για ρεαλιστικά προβλήματα, χώρος αναζήτησης και ο απαιτούμενος για την εύρεση της λύσης χρόνος, μπορούν να χρησιμοποιηθούν αλγόριθμοι που εκμεταλλεύονται ευρετικούς μηχανισμούς, δηλαδή στρατηγικές (συνήθως συναρτήσεις που εξαρτώνται από το εκάστοτε πρόβλημα) οι οποίες αξιολογούν προσεγγιστικά τις ενδιάμεσες καταστάσεις ως προς την εκτιμώμενη απόσταση τους από μία τελική κατάσταση, επεκτείνουν πρώτα αυτές με τη βέλτιστη ευρετική τιμή (οι οποίες αναμένεται να οδηγήσουν συντομότερα σε λύση) ή/και κλαδεύουν τις υπόλοιπες. Οι ευρετικοί μηχανισμοί δεν είναι αντικειμενικοί και, παρόλο που κωδικοποιούνται αλγοριθμικά υπό τη μορφή της ευρετικής συνάρτησης, δεν μπορούν να θεωρηθούν αλγόριθμοι. Αυτό γιατί, προκειμένου να μειώσουν το χώρο αναζήτησης ή να επιταχύνουν την εύρεση της λύσης, λειτουργούν προσεγγιστικά και «διαισθητικά» (περίπου όπως οι άνθρωποι), ενώ οι αλγόριθμοι είναι ακριβείς και λειτουργούν πάντα ορθά. Στην πλειονότητα των περιπτώσεων πάντως οι ευρετικές στρατηγικές οδηγούν σε πολύ καλά αποτελέσματα (αναλόγως βέβαια του προβλήματος), ωστόσο απέχουν πολύ από το να προσομοιώνουν τους μηχανισμούς της ανθρώπινης σκέψης: η τελευταία χρησιμοποιεί επίσης ευρετικές μεθόδους οι οποίες όμως είναι ποιοτικές, όχι ποσοτικές / αριθμητικές όπως η ευρετική συνάρτηση, και φαίνεται να αποδίδουν καλύτερα.

Ένας βασικός ευρετικός αλγόριθμος είναι ο HC (Hill Climbing ή αναρρίχηση λόφων), ο οποίος μοιάζει με τον DFS αλλά σε κάθε επανάληψη κλαδεύει όλες τις καταστάσεις που προκύπτουν από μία επέκταση εκτός από την ευρετικά βέλτιστη (δηλαδή κάθε στιγμή το Μ.Α. έχει μία κατάσταση) και μεταβαίνει στην τελευταία μόνο αν έχει καλύτερη ευρετική τιμή από το γονέα της· διαφορετικά τερματίζει έχοντας βρει μία τοπικά βέλτιστη λύση. Προφανώς ο HC δεν είναι πλήρης αλλά είναι πολύ γρήγορος και καθόλου μνημοβόρος. Υπάρχουν διάφορες παραλλαγές του που θυσιάζουν λίγη από την ταχύτητα του προκειμένου να αυξήσουν την πιθανότητα του να βρει λύση. Μία παραλλαγή είναι ο EHC (Enforced Hill Climbing εξαναγκασμένη αναρρίχηση λόφων), στον οποίον διατηρούνται στο Μ.Α. τα αδέρφια του τρέχοντος κόμβου και, αν η επέκταση του τελευταίου δεν οδηγήσει σε μετάβαση, αντί ο αλγόριθμος να τερματίσει εκτελεί μία αναζήτηση κατά πλάτος στα αδέρφια του μέχρι να βρεθεί μία καλύτερη κατάσταση οπότε και συνεχίζεται η αναρρίχηση από εκεί. Επίσης δημοφιλής είναι και ο SA (Simulated Annealing ή προσομοιωμένη ανόπτηση), ο οποίος δίνει μία πιθανότητα μετάβασης σε χειρότερες καταστάσεις (p), αφήνοντας έτσι περιθώριο στην αναζήτηση να ξεφύγει από τοπικά βέλτιστα. Αν η πιθανότητα p τείνει στο 0 ο SA λειτουργεί όπως ο HC. Επίσης υπάρχει ο TS (Taboo Search ή αναζήτηση με απαγορεύσεις), όπου σε κάθε επέκταση γίνεται πάντα μετάβαση στο καλύτερο παιδί, ακόμα και αν είναι χειρότερη κατάσταση από την τρέχουσα, και η αναζήτηση συμβουλεύεται μία λίστα απαγορευμένων καταστάσεων (παρόμοιας λειτουργικότητας με το Κλειστό Σύνολο αλλά σταθερού μεγέθους). Ο BS (Beam Search ή ακτινωτή αναζήτηση), όπου ένας σταθερός αριθμός εκ των καλύτερων καταστάσεων παραμένει στο Μ.Α. δίνοντας τη δυνατότητα οπισθοδρόμησης αν χρειαστεί, είναι μία ακόμα επέκταση του κεντρικού αλγορίθμου αναρρίχησης λόφων.

Όλοι αυτοί οι ευρετικοί αλγόριθμοι κατάγονται από τη θεωρία μαθηματικής βελτιστοποίησης, όπου αναπτύχθηκαν για να εντοπίζουν το ελάχιστο ή το μέγιστο μίας πραγματική συνάρτησης διακριτής μεταβλητής. Στην επίλυση προβλημάτων τον ρόλο της τελευταίας προφανώς τον παίζει η ευρετική συνάρτηση και ο χώρος των λύσεων οπτικοποιείται ως ένα γεωγραφικό «τοπίο»: όσο περισσότερο δύο λύσεις διαφέρουν τόσο απέχουν μεταξύ τους σε αυτό το τοπίο, ενώ όσο καλύτερη ευρετική τιμή έχει μία λύση τόσο υψηλότερα από το επίπεδο του «εδάφους» τοποθετείται σε αυτό το τοπίο. Το τελευταίο, καθώς οι υποψήφιες καταστάσεις είναι διακριτές μεταξύ τους, ουσιαστικά είναι ένας γράφος με κορυφές τις καταστάσεις και ακμές τους τελεστές μετάβασης. Η παραλλαγή της αναρρίχησης λόφων σε συνεχή χώρο, με στόχο την εύρεση ακρότατου μιας συνάρτησης συνεχούς μεταβλητής, ονομάζεται άνοδος κλίσης (gradient ascent, αν η συνάρτηση εκφράζει βελτιστότητα και αναζητείται το μέγιστό της) ή κάθοδος κλίσης (gradient descent, αν η συνάρτηση εκφράζει σφάλμα / απόκλιση από το βέλτιστο και αναζητείται το ελάχιστο της) και υλοποιείται με μεθόδους του απειροστικού λογισμού.

Άλλος δημοφιλής ευρετικός αλγόριθμος είναι ο BestFS (αναζήτηση πρώτα στο καλύτερο) ο οποίος κρατά όλες τις καταστάσεις στο Μ.Α. και μοιάζει με τον BFS, μόνο που σε κάθε επέκταση εφαρμόζει τον ευρετικό μηχανισμό και στην επόμενη επανάληψη μεταβαίνει στο ευρετικά βέλτιστο

παιδί. Είναι πλήρης, μνημοβόρος και δεν εγγυάται ότι θα βρει τη βέλτιστη λύση αφού εξαρτάται απόλυτα από την εγκυρότητα των εκτιμήσεων της ευρετικής συνάρτησης. Τροποποίηση του BestFS αποτελεί ο πλήρης και βέλτιστος αλγόριθμος A*, στον οποίον η ευρετική τιμή που αντιστοιχίζεται σε κάθε νέα κατάσταση k για να την αξιολογήσει ο μηχανισμός δεν είναι μόνο μία εκτίμηση A της απόστασης της από μία τελική κατάσταση, αλλά το άθροισμα A συν την ακριβή απόσταση της k από τη ρίζα. Ο A* εγγυάται ότι θα βρει τη βέλτιστη λύση αρκεί η ευρετική συνάρτηση να είναι πάντα υποεκτίμηση της πραγματικής απόστασης από τη λύση και ποτέ υπερεκτίμηση («αποδεκτή συνάρτηση»). Σε περίπτωση που είναι σπουδαιότερη η ταχύτητα παρά η βελτιστότητα δε χρειάζεται η ευρετική συνάρτηση να είναι αποδεκτή. Παραλλαγή του A* αποτελεί ο IDA* (A* με επαναληπτική εκβάθυνση) ο οποίος αναπτύσσει το δένδρο αναζήτησης κατά βάθος σε διαδοχικές επαναλήψεις, αξιοποιώντας την ευρετική συνάρτηση του A* για να επιλέξει την επεκτεινόμενη κάθε φορά κατάσταση, αλλά όταν η ευρετική τιμή μίας νέας κατάστασης ξεπερνά το όριο που έχει τεθεί για την τρέχουσα επανάληψη όλο το υποδένδρο το οποίο ξεκινά από αυτήν κλαδεύεται. Στην επόμενη επανάληψη, όπου όπως στον ID το δένδρο αναζήτησης κατασκευάζεται από την αρχή, το νέο ευρετικό όριο τίθεται στη μικρότερη τιμή που εμφανίστηκε στις καταστάσεις οι οποίες κλαδεύτηκαν κατά την προηγούμενη επανάληψη.

Εξελικτικός υπολογισμός

Σε προβλήματα βελτιστοποίησης μπορεί εναλλακτικά να χρησιμοποιηθεί κάποιος τύπους εξελικτικού υπολογισμού όπως οι εξελικτικοί αλγόριθμοι. Ο εξελικτικός υπολογισμός είναι μία κατηγορία εργαλείων της υπολογιστικής νοημοσύνης ο οποίος βασίζεται στην ιδέα της σταδιακής ανάπτυξης, με μια επαναληπτική διαδικασία, πιθανών λύσεων για ένα πρόβλημα μέσω πολλαπλών παράλληλων αναζητήσεων στο χώρο των λύσεων. Η διαφορά από άλλες μεθόδους βελτιστοποίησης είναι ότι οι υποψήφιες λύσεις αλληλεπιδρούν και αλληλεπηρεάζονται ώσπου να τερματίσει η διαδικασία. Όταν αυτή η αλληλεπίδραση συμβαίνει με βάση τις αρχές της βιολογικής εξέλιξης των ειδών τότε μιλάμε για εξελικτικούς αλγορίθμους. Στον τομέα της βέλτιστης επίλυσης προβλημάτων συνήθως χρησιμοποιούνται οι γενετικοί αλγόριθμοι, μία υποκατηγορία εξελικτικών αλγορίθμων η οποία μοιάζει με την αναρρίχηση λόφων αλλά δεν παγιδεύεται εύκολα σε τοπικά βέλτιστες λύσεις.

Στους γενετικούς αλγορίθμους αρχικά παράγεται ένα σύνολο N υποψήφιων λύσεων για το εκάστοτε πρόβλημα (πληθυσμός n), το οποίο κατασκευάζεται τυχαία και επομένως τα περισσότερα μέλη του είναι άκυρα ή μη βέλτιστα ως λύσεις. Αυτοί οι υποψήφιοι αξιολογούνται με μία καθοριζόμενη από τον χειριστή (συνήθως ευρευτική) πραγματική συνάρτηση διακριτής μεταβλητής, τη συνάρτηση καταλληλότητας, η οποία επιχειρεί να βαθμολογήσει κάθε υποψήφιο ανάλογα με το πόσο κοντά βρίσκεται σε κάποια ιδανική λύση. Ακολούθως από τον αρχικό πληθυσμό σχηματίζονται N/2 ζεύγη υποψηφίων («γονέων»), με μεγαλύτερη προτεραιότητα στις πιο θετικά αξιολογημένες λύσεις, όπου κάθε υποψήφιος μπορεί να συμμετέχει σε περισσότερα από ένα ζεύγη. Τα μέλη κάθε ζεύγους συνδυάζονται με κάποιον

τρόπο μεταξύ τους και το αποτέλεσμα είναι δύο νέες υποψήφιες λύσεις («απόγονοι»). Ο νέος πληθυσμός n+1 αποτελείται από το σύνολο αυτών των απογόνων (πλήρης ανανέωση). Εναλλακτικά οι απόγονοι μπορούν να συνυπάρχουν με μέλη του αμέσως προηγούμενου πληθυσμού n (μερική ανανέωση), σε κάθε περίπτωση όμως ο πληθάριθμος N παραμένει σταθερός σε κάθε «γενιά».

Το ποσοστό υποψηφίων που αντικαθίσταται από απογόνους ονομάζεται «χάσμα γενεών» και στην πλήρη ανανέωση είναι 100%, ενώ στη μερική ανανέωση η πιθανότητα αντικατάστασης μίας λύσης της γενιάς n από απόγονο της γενιάς n+1 είναι αντιστρόφως ανάλογη της καταλληλότητας της. Η διαδικασία αυτή επαναλαμβάνεται μέχρι να ικανοποιηθεί κάποιο κριτήριο τερματισμού, δηλαδή συνήθως να βρεθεί μία λύση που αξιολογείται ως βέλτιστη από τη συνάρτηση καταλληλότητας ή ο μέσος όρος των λύσεων του τρέχοντος πληθυσμού να τείνει να συγκλίνει σε μία μόνο λύση (ή μικρές παραλλαγές μίας). Αυτή η μεθοδολογία επιχειρεί να μιμηθεί τη βιολογική ιδέα της γενετικής διαφοροποίησης και της φυσικής επιλογής, των πυλώνων της εξέλιξης των ειδών, αλλά ουσιαστικώς η φυσική επιλογή αντικαθίσταται από μία τεχνητή επιλογή η οποία γίνεται μέσω της συνάρτησης καταλληλότητας. Η τελευταία αποτελεί και το υπέρτατο κριτήριο για την πραγματική απόδοση του αλγορίθμου.

1.2 Αντικείμενο Διπλωματικής

Η διπλωματική έχει ως στόχο την εξέταση της απόδοσης του αλγορίθμου Τοπικής Ακτινικής Αναζήτησης(Local Beam search), ενός ευριστικού αλγορίθμου τοπικής αναζήτησης, για την εύρεση λύσεων σε προβλήματα ικανοποίησης περιορισμών. Αυτό θα επιτευχθεί με την υλοποίηση του αλγορίθμου και την επανειλημμένη εκτέλεση του για πολλά διαφορετικά προβλήματα. Επίσης, θα υλοποιήσουμε τον αλγόριθμο Αναρρίχησης λόφων (Hill climbing) με τυχαίες επανεκκινήσεις (που είναι γνωστός ως Min-conflicts, Solving Large Scale Constraint Satisfaction and Scheduling Problems using a Heuristic Repair Method – Steven Minton,Mark D.Johnston,Andrew B. Philips, Philip Laird) στα ίδια προβλήματα με στόχο την σύγκριση των δύο αλγορίθμων .

Οι δύο αλγόριθμοι έχουν πολλά κοινά, αλλά στην ουσία ο αλγόριθμος Τοπικής Ακτινικής Αναζήτησης αποτελεί πιο εξελιγμένη μορφή, αφού συγκρατεί, στη μνήμη, περισσότερες της μίας κατάστασης μέσα στις επαναλήψεις. Το θέμα είναι αν επιτυγχάνει καλύτερη απόδοση στην πράξη και για όλα τα είδη προβλημάτων η αν αποτελεί απλά μια χρονοβόρα έκδοση του αλγορίθμου Αναρρίχησης Λόφων. Θα ήταν ενδιαφέρον να προσδιορίσουμε σε ποια προβλήματα πρέπει να επιλέξουμε τον αλγόριθμο της τοπικής ακτινικής αναζήτησης και σε ποια τον αλγόριθμο Ελάχιστων Συγκρούσεων με τυχαίες επανεκκινήσεις, ώστε να έχουμε τη καλύτερη λύση στο καλύτερο χρόνο με τις λιγότερες επαναλήψεις.

Αυτή η διπλωματική έχει ως σκοπό την απάντηση αυτών των ερωτημάτων, μέσω της σύγκρισης των δύο ευρετικών αλγορίθμων τοπικής

αναζήτησης, του αλγορίθμου Αναρρίχησης λόφων και του αλγορίθμου Ακτινικής Τοπικής Αναζήτησης, τους οποίους θα αναλύσουμε στο παρακάτω κεφάλαιο.

2. ΘΕΩΡΗΤΙΚΟ ΥΠΟΒΑΘΡΟ

Όπως προαναφέρθηκε η διπλωματική ασχολείται με τον αλγόριθμο Ακτινικής Τοπικής Αναζήτησης σε προβλήματα ικανοποίησης περιορισμών. Οπότε σε αυτό το κεφάλαιο θα αναλύσουμε τα προβλήματα αυτά και τη λειτουργία της τοπικής αναζήτησης, αναλύοντας τον αλγόριθμο Ακτινικής Τοπικής Αναζήτησης και τον αλγόριθμο Αναζήτησης με Αναρρίχηση Λόφων, τον οποίο θα χρησιμοποιήσουμε ως μέτρο σύγκρισης του πρώτου στα επόμενα κεφάλαια.

2.1 Τοπική Αναζήτηση

Οι αλγόριθμοι Τυφλής αναζήτησης και ο ευριστικός αλγόριθμος Α* που είδαμε στην εισαγωγή εξερευνούν χώρους αναζήτησης συστηματικά και διατηρούν μια ή περισσότερες διαδρομές στη μνήμη κι όταν βρεθεί μια κατάσταση στόχου, η διαδρομή προς αυτή τη κατάσταση είναι λύση. Σε πολλά προβλήματα βελτιστοποίησης η διαδρομή ως τη κατάσταση στόχου δεν έχει σημασία. Το μόνο που μας ενδιαφέρει είναι να βρούμε μια κατάσταση στόχου και να προσδιορίσουμε τη μορφή της.

Σε τέτοιες περιπτώσεις (και όχι μόνο) μπορούμε να χρησιμοποιήσουμε αλγόριθμους που δεν ενδιαφέρονται για διαδρομές και βασίζονται στην επαναληπτική βελτίωση (iterative improvement). Δηλαδή ξεκινάμε με μια αρχική κατάσταση και προσπαθούμε να τη βελτιώνουμε.

Οι αλγόριθμοι που θα περιγράψουμε ονομάζονται αλγόριθμοι τοπικής αναζήτησης (local search) και συνήθως λειτουργούν χρησιμοποιώντας μια μόνο (τρέχουσα) κατάσταση (αντί για πολλαπλά μονοπάτια) και γενικά μετακινούνται μόνο σε γειτονικές τους καταστάσεις.

Οι διαδρομές που ακολουθούνται από την αναζήτηση δεν διατηρούνται στην μνήμη. Σε κάθε βήμα ενός αλγόριθμου τοπικής αναζήτησης έχουμε μια πλήρη αλλά ατελή “λύση” του προβλήματος.

Δύο βασικά πλεονεκτήματα των αλγορίθμων τοπικής αναζήτησης είναι:

1. Χρησιμοποιούν πολύ λίγη μνήμη
2. Μπορούν να βρίσκουν λογικές λύσεις σε μεγάλους ή και άπειρους συνεχείς χώρους καταστάσεων

Οι αλγόριθμοι τοπικής αναζήτησης είναι, επίσης, χρήσιμοι για την επίλυση αμιγών προβλημάτων βελτιστοποίησης, στα οποία ο σκοπός είναι να βρεθεί η καλύτερη κατάσταση, σύμφωνα με την αντικειμενική συνάρτηση.

Για να γίνει πιο κατανοητή η τοπική αναζήτηση είναι πολύ χρήσιμο να εξετάσουμε το τοπίο του χώρου καταστάσεων. Ένα τοπίο έχει «τοποθεσία» (που ορίζεται από την κατάσταση) και «υψόμετρο» (που ορίζεται από τη τιμή της ευρετικής συνάρτησης κόστους ή της αντικειμενικής συνάρτησης). Αν το υψόμετρο αντιστοιχεί στο κόστος, τότε ο σκοπός είναι να βρεθεί η χαμηλότερη κοιλάδα, δηλαδή ένα καθολικό ελάχιστο, αν αντιστοιχεί σε μια αντικειμενική συνάρτηση, τότε ο σκοπός είναι να βρεθεί η ψηλότερη κορυφή, δηλαδή ένα καθολικό μέγιστο. Ένας αλγόριθμος τοπικής αναζήτησης βρίσκει πάντα μία κατάσταση στόχου, αν υπάρχει και ένας βέλτιστος βρίσκει πάντα ένα καθολικό ελάχιστο ή μέγιστο.

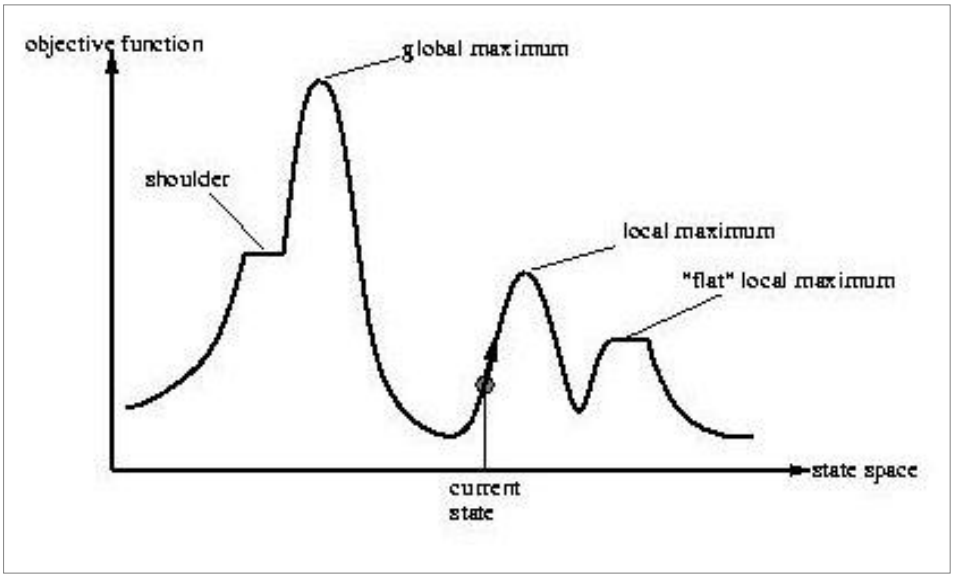
Οι βασικοί αλγόριθμοι τοπικής αναζήτησης είναι οι εξής:

- Αναζήτηση με Αναρρίχηση Λόφων
- Προσομοιωμένη Ανόπτηση
- Τοπική ακτινική αναζήτηση
- Γενετικοί Αλγόριθμοι

Και οι τέσσερις έχουν προαναφερθεί στην εισαγωγή. Σε αυτό το κεφάλαιο θα αναλύσουμε εκτενέστερα την Αναζήτηση με Αναρρίχηση Λόφων και την Τοπική ακτινική αναζήτηση με τις οποίες θα ασχοληθούμε στην συνέχεια της διπλωματικής.

2.1.1 ΑΝΑΖΗΤΗΣΗ ΜΕ ΑΝΑΡΡΙΧΗΣΗ ΛΟΦΩΝ(Hill climbing)

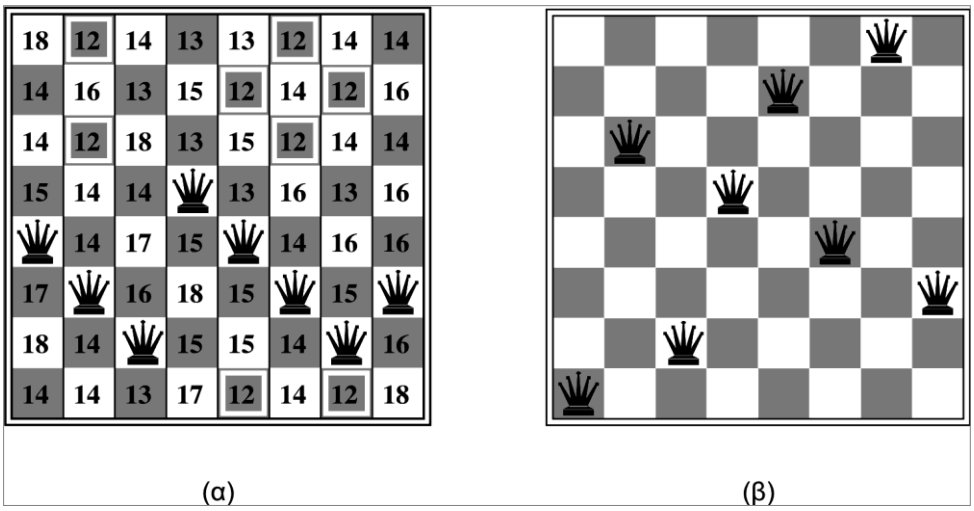
Ο αλγόριθμος hill climbing αποτελεί τον απλούστερο αλγόριθμο τοπικής αναζήτησης. Είναι απλώς ένας βρόχος που μετακινείται συνεχώς προς την κατεύθυνση της αυξανόμενης τιμής, όπως βλέπουμε και στο παρακάτω σχήμα. Ο αλγόριθμος τερματίζει όταν φτάσει στην κορυφή όπου κανένας γειτονικός κόμβος δεν έχει υψηλότερη τιμή. Δεν κοιτάζει πιο πέρα από τους άμεσους γείτονες της τρέχουσας κατάστασης.



Εικόνα 1

Για να επιδείξουμε την αναρρίχηση λόφων, θα χρησιμοποιήσουμε το πρόβλημα των 8 βασιλισσών. Κλασικό παράδειγμα προβλήματος περιορισμών, το οποίο απαιτεί να τοποθετηθούν 8 βασίλισσες σε μία σκακιέρα 8x8, χωρίς να απειλούν η μία την άλλη. Το πρόβλημα ορίζεται και για περισσότερες των 8 βασιλισσών. Η δυσκολία στην επίλυσή του αυξάνει εκθετικά. Χρησιμοποιείται για τη μέτρηση της απόδοσης αλγορίθμων ικανοποίησης περιορισμών.

Οι αλγόριθμοι τοπικής αναζήτησης κανονικά χρησιμοποιούν μια διατύπωση με πλήρεις καταστάσεις, όπου κάθε κατάσταση έχει 8 βασίλισσες στη σκακιέρα, μια σε κάθε στήλη. Η συνάρτηση διαδοχών επιστρέφει όλες τις δυνατές καταστάσεις που παράγονται από την μετακίνηση της μιας μόνο βασίλισσας σε άλλο τετράγωνο της ίδιας στήλης. Η ευρετική συνάρτηση κόστους h είναι ο αριθμός των ζευγών βασιλισσών που αλληλεπιδρούν, είτε άμεσα, είτε έμμεσα. Το καθολικό ελάχιστο είναι το μηδέν. Η εικόνα δείχνει μια κατάσταση για h=12. Επίσης, δείχνει τις τιμές όλων των διαδόχων της, όπου οι καλύτεροι έχουν h=12. Οι αλγόριθμοι αναρρίχησης λόφων, κανονικά, διαλέγουν τυχαία από το σύνολο των καλύτερων διαδόχων, αν υπάρχουν περισσότεροι από ένας.



Εικόνα 2

Για καλύτερη κατανόηση παραθέτουμε παρακάτω τον αλγόριθμο σε βήματα και στη συνέχεια σε ψευδοκώδικα την έκδοση, της πλέον απότομης ανάβασης, της πιο βασικής τεχνικής τοπικής αναζήτησης.

1. Η αρχική κατάσταση είναι η τρέχουσα κατάσταση.
2. Αν η κατάσταση είναι μια τελική τότε ανέφερε την λύση και σταμάτησε.
3. Βρες τις γειτονικές καταστάσεις
4. Επέλεξε την καλύτερη (κόμβο με μεγαλύτερη τιμή VALUE).
5. Επέστρεψε στο βήμα 2.

```
function Hill_Climbing (problem, Eval-fn)
returns a state that is a local maximum
inputs: problem, a search problem
        Eval-fn, an evaluation function
local variables: current, next, nodes
                 neighbors, a set of nodes
        current ← MakeNode(InitialState[problem])
        loop do
                neighbors ← SuccessorStates(current)
                if neighbors = 0 return current
                Eval-fn(neighbors)
                next ← a highest-valued node of neighbors
                if the value of next is less or equal to the value of current
                        return current
        end
```

Άπληστη Τοπική Αναζήτηση

Η αναρρίχηση λόφων λέγεται και άπληστη τοπική αναρρίχηση, επειδή αρπάζει μία γειτονική κατάσταση χωρίς να σκεφτεί παραπέρα που θα πάει στη συνέχεια. Οι άπληστοι αλγόριθμοι συχνά αποδίδουν πολύ καλά. Η αναρρίχηση λόφων συχνά κάνει μεγάλη πρόοδο προς μια λύση, επειδή μπορεί να βελτιώσει μια κακή κατάσταση. Δυστυχώς, ο αλγόριθμος συχνά παγιδεύεται για τους παρακάτω λόγους:

Τοπικά Βέλτιστα (local optima)
Η αναζήτηση μπορεί να κολλήσει σε μια κατάσταση της οποίας όλες οι γειτονικές έχουν χειρότερο κόστος, αλλά δεν είναι η βέλτιστη.

Οροπέδια (plateaus)
Αν όλα τα γειτονικά σημεία έχουν το ίδιο κόστος η επιλογή γίνεται τυχαία.

Κορυφογραμμές (ridges)
Οι κορυφογραμμές είναι ψηλά και εντοπίζονται εύκολα, όμως από εκεί και πέρα η αναζήτηση είναι πολύ αργή γιατί αποτελούνται από μια ακολουθία τοπικών βέλτιστων.

Σε αυτές τις περιπτώσεις ο αλγόριθμος φτάνει σε ένα σημείο από το οποίο δεν μπορεί να κάνει καμία πρόοδο.

Ο αλγόριθμος της πλέον απότομης ανάβασης όταν φτάσει σε ένα οροπέδιο, όπου η καλύτερη διαδοχική τιμή έχει την ίδια τιμή με την τρέχουσα κατάσταση, σταματάει. Μια βελτίωση του αλγορίθμου θα ήταν να επιτρέπονται οι πλάγιες κινήσεις με την ελπίδα ότι το οροπέδιο είναι ώμος όπως είδαμε στην εικόνα 1. Όμως, στη περίπτωση που το οροπέδιο αποτελεί ένα επίπεδο τοπικό μέγιστο, που δεν είναι ώμος, ο αλγόριθμος θα εγκλωβίζεται σε έναν ατέρμονα βρόχο. Οπότε μια συνηθισμένη λύση είναι να τίθεται ένα όριο στον αριθμό των διαδοχικών πλάγιων κινήσεων που επιτρέπονται.

Στοχαστική αναρρίχηση Λόφων

Έχουν επινοηθεί πολλές παραλλαγές της αναρρίχησης λόφων. Η στοχαστική αναρρίχηση λόφων διαλέγει τυχαία από τις διαθέσιμες κινήσεις προς τα επάνω. Η πιθανότητα επιλογής μπορεί να διαφέρει ανάλογα με το πόσο απότομη κλήση έχει κάθε τέτοια κίνηση. Η παραλλαγή αυτή συνήθως συγκλίνει πιο αργά από την μέθοδο της πλέον απότομης ανάβασης, αλλά σε μερικά τοπία καταστάσεων βρίσκει καλύτερες λύσεις.

Αναρρίχηση λόφων με την πρώτη επιλογή

Η αναρρίχηση λόφων με την πρώτη επιλογή υλοποιεί τη στοχαστική αναρρίχηση λόφων παράγοντας διαδοχικές καταστάσεις τυχαία, μέχρι να παραχθεί κάποια που είναι καλύτερη από την τρέχουσα κατάσταση. Η στρατηγική αυτή είναι καλή όταν μια κατάσταση έχει πολλές διαδοχικές καταστάσεις.

Αναρρίχηση λόφων με τυχαίες επανεκκινήσεις

Οι αλγόριθμοι αναρρίχησης λόφων που περιγράψαμε μέχρι εδώ είναι μη πλήρεις, συχνά δεν καταφέρνουν να βρουν μια κατάσταση στόχου ενώ υπάρχει, επειδή μπορεί να παγιδευτούν σε τοπικά μέγιστα. Η αναρρίχηση λόφων με τυχαίες επανεκκινήσεις αποτελεί ένα πλήρη αλγόριθμο με πιθανότητα που αγγίζει το 1. Πραγματοποιεί μια σειρά αναζητήσεων αναρρίχησης λόφων από τυχαία παραγόμενες αρχικές καταστάσεις, σταματώντας όταν βρεθεί μια κατάσταση στόχου. Αν κάθε αναρρίχηση λόφων έχει πιθανότητα επιτυχίας, ο αναμενόμενος αριθμός βημάτων είναι ίσος με 1/p.

Η επιτυχία του αλγορίθμου εξαρτάται σε μεγάλο βαθμό από το σχήμα του τοπίου του χώρου καταστάσεων. Αν υπάρχουν λίγα τοπικά μέγιστα και οροπέδια ο αλγόριθμος αναρρίχησης λόφων με τυχαίες επανεκκινήσεις θα βρει μια καλή λύση πολύ γρήγορα. Πολλά όμως πραγματικά προβλήματα έχουν ένα τοπίο με εκθετικά μεγάλο αριθμό τοπικών μεγίστων στα οποία μπορεί να παγιδευτεί, αλλά παρ'όλα αυτά ένα πολύ καλό τοπικό μέγιστο συχνά μπορεί να βρεθεί μετά από μικρό αριθμό επανεκκινήσεων.

2.1.2 ΤΟΠΙΚΗ ΑΚΤΙΝΙΚΗ ΑΝΑΖΗΤΗΣΗ

Ο αλγόριθμος ο οποίος θα μας απασχολήσει στα παρακάτω κεφάλαια είναι ο αλγόριθμος της τοπικής ακτινικής αναζήτησης. Ο οποίος είναι μια προσαρμογή της ακτινικής αναζήτησης, ενός αλγορίθμου που βασίζεται στη διαδρομή. Ο αλγόριθμος παρακολουθεί k καταστάσεις αντί μόνο μία. Ξεκινά με k τυχαία παραγόμενες καταστάσεις. Σε κάθε βήμα παράγονται όλοι οι διάδοχοι και των k καταστάσεων. Αν οποιαδήποτε από αυτές ικανοποιεί τον στόχο τότε ο αλγόριθμος σταματά. Αλλιώς, επιλέγει τους k καλύτερους διαδόχους από όλη τη λίστα και επαναλαμβάνει.

Με την πρώτη ματιά, μια τοπική ακτινική αναζήτηση με k καταστάσεις ίσως φαίνεται να μην είναι τίποτα περισσότερο από την εκτέλεση k τυχαίων επανεκκινήσεων παράλληλα και όχι στη σειρά. Στη πραγματικότητα, οι δύο αλγόριθμοι είναι εντελώς διαφορετικοί. Αυτός είναι ο σκοπός της διπλωματικής να αποδείξουμε ότι στην πράξη η τοπική ακτινική αναζήτηση είναι πιο αποδοτική. Στην αναζήτηση με τυχαίες επανεκκινήσεις, η διαδικασία εκτελείται ανεξάρτητα από τις άλλες. Σε μια τοπική ακτινική αναζήτηση, μεταβιβάζονται χρήσιμες πληροφορίες μεταξύ των k παράλληλων νημάτων αναζήτησης οπότε οι αλγόριθμοι δεν έχουν την ίδια απόδοση.

Στοχαστική ακτινική αναζήτηση

Η τοπική ακτινική αναζήτηση μπορεί να πάσχει από έλλειψη ποικιλίας μεταξύ των k καταστάσεων οι αναζητήσεις μπορεί σύντομα να συγκεντρωθούν σε μια μικρή περιοχή του χώρου καταστάσεων, κάνοντας την αναζήτηση ελάχιστα καλύτερη από μια δαπανηρή έκδοση της αναρρίχησης λόφων. Μια παραλλαγή της τοπικής ακτινικής αναζήτησης είναι η στοχαστική ακτινική αναζήτηση, ανάλογη της στοχαστικής αναζήτησης αναρρίχησης λόφων, η οποία βοηθά να αμβλυνθεί αυτό το πρόβλημα. Αντί να διαλέγει τους k καλύτερους, διαλέγει k διαδοχικούς τυχαία, όπου η πιθανότητα εκλογής ενός δεδομένου διαδόχου είναι αύξουσα συνάρτηση της αξίας του.

2.2 Προβλήματα Ικανοποίησης Περιορισμών

Αναφέραμε ότι τα προβλήματα μπορεί να λύνονται με αναζήτηση σε έναν χώρο καταστάσεων. Αυτές οι καταστάσεις μπορούν να αξιολογούνται με ευριστικούς μηχανισμούς, ειδικούς για το συγκεκριμένο πεδίο, και να εξετάζονται για να διαπιστωθεί αν είναι καταστάσεις στόχου. Από άποψη του αλγορίθμου αναζήτησης κάθε κατάσταση είναι ένα μαύρο κουτί χωρίς ευδιάκριτη εσωτερική δομή.

Σε αυτό το κεφάλαιο εξετάζονται προβλήματα ικανοποίησης περιορισμών που οι καταστάσεις τους και ο έλεγχος στόχου τους συμμορφώνονται με μια καθιερωμένη, δομημένη και πολύ απλή αναπαράσταση. Μπορούν να οριστούν αλγόριθμοι αναζήτησης που εκμεταλλεύονται τη δομή των καταστάσεων και χρησιμοποιούν ευρετικούς μηχανισμούς γενικής χρήσης και ειδικούς για το συγκεκριμένο πρόβλημα, ώστε να γίνει δυνατή η επίλυση μεγάλων προβλημάτων. Ίσως το σπουδαιότερο είναι ότι η καθιερωμένη αναπαράσταση του ελέγχου στόχου αποκαλύπτει τη δομή του ίδιου του προβλήματος. Αυτό οδηγεί σε μεθόδους αποσύνθεσης των προβλημάτων και στη κατανόηση της βαθύτερης σχέσης μεταξύ της δομής ενός προβλήματος και της δυσκολίας επίλυσης του.

Ένα πρόβλημα ικανοποίησης περιορισμών ορίζεται από ένα σύνολο μεταβλητών X1 ,X2 , …, Xn και ένα σύνολο περιορισμών C1 , C2 , …, Cm. Κάθε μεταβλητή έχει ένα πεδίο Di των δυνατών τιμών της. Κάθε περιορισμός Ck αναφέρεται σε κάποιο υποσύνολο των μεταβλητών και καθορίζει τους

επιτρεπτούς συνδυασμούς τιμών και γι'αυτό το υποσύνολο. Μια κατάσταση του προβλήματος ορίζεται με ανάθεση τιμών σε μερικές ή σε όλες τις μεταβλητές, {Xi = vi , Xj = vj , …}. Μια ανάθεση τιμής που δεν παραβιάζει κανέναν περιορισμό ονομάζεται συνεπής ή νόμιμη ανάθεση. Πλήρης ανάθεση είναι μια ανάθεση που περιλαμβάνει όλες τις μεταβλητές, και λύση ενός προβλήματος ικανοποίησης περιορισμών είναι η πλήρης ανάθεση τιμών που ικανοποιεί όλους τους περιορισμούς. Μερικά προβλήματα ικανοποίησης περιορισμών απαιτούν, επίσης, μια λύση που μεγιστοποιεί την αντικειμενική συνάρτηση.

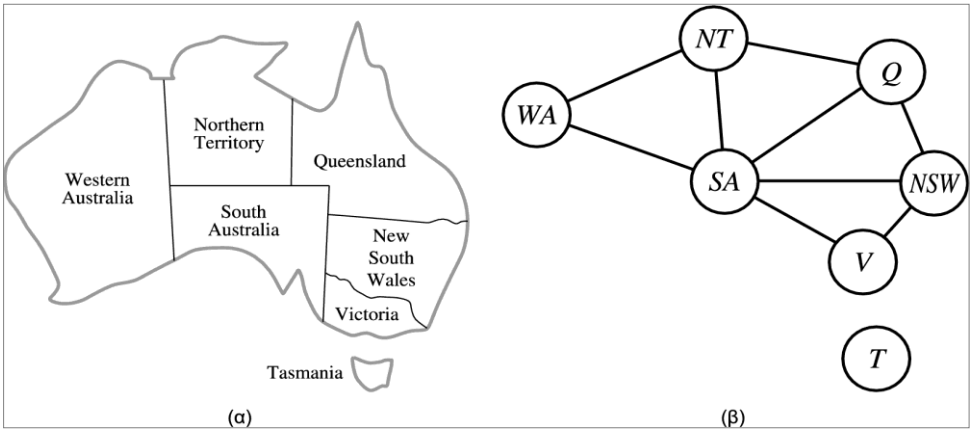
Είναι χρήσιμο να αντιλαμβανόμαστε, ένα πρόβλημα ικανοποίησης περιορισμών σαν γράφημα περιορισμών όπως η εικόνα 3β. Οι κόμβοι του γραφήματος αντιστοιχούν σε μεταβλητές και τα τόξα σε περιορισμούς.

Στην εικόνα 3α βλέπουμε τις κύριες πολιτείες τις Αυστραλίας. Ο χρωματισμός αυτού του χάρτη μπορεί να θεωρηθεί ως πρόβλημα ικανοποίησης περιορισμών. Ο στόχος είναι να αποδοθούν χρώματα στη περιοχή έτσι ώστε καμία γειτονική περιοχή να μην έχει το ίδιο χρώμα.

Στο πρόβλημα αυτό έχουμε:

Μεταβλητές: *WA*, *NT*, *Q*, *NSW*, *V*, *SA*, *T*
Πεδία ορισμού: {*κόκκινο*, *πράσινο*, *μπλε*} για όλες τις μεταβλητές
Περιορισμοί: Να μην έχουν οι γειτονικές περιοχές ίδιο χρώμα
Εναλλακτικές αναπαραστάσεις:

- Απαρίθμηση επιτρεπτών συνδυασμών
  - { (*κόκκινο*, *πράσινο*), (*κόκκινο*, *μπλε*), (*πράσινο*, *κόκκινο*), (*πράσινο*, *μπλε*), (*μπλε*, *κόκκινο*), (*μπλε*, *πράσινο*) }
- Ειδική σημειογραφία
  - *WA* ≠ *NT*



Εικόνα 3

Η αντιμετώπιση του προβλήματος, ως πρόβλημα ικανοποίησης περιορισμών έχει πολλά σημαντικά πλεονεκτήματα. Επειδή η αναπαράσταση των καταστάσεων σε ένα πρόβλημα ικανοποίησης περιορισμών γίνεται σύμφωνα με ένα καθιερωμένο πρότυπο -με ένα σύνολο μεταβλητών στις οποίες δίνονται τιμές- η συνάρτηση διαδοχών και ο έλεγχος στόχου μπορούν

να γράφονται με ένα γενικό τρόπο που εφαρμόζεται σε όλα τα προβλήματα ικανοποίησης περιορισμών. Επίσης, μπορούμε να αναπτύσσουμε αποτελεσματικούς ευρετικούς μηχανισμούς γενικής χρήσης που δεν απαιτούν καμία πρόσθετη ειδική γνώση του συγκεκριμένου πεδίου. Τέλος, η δομή του γραφήματος περιορισμών μπορεί να χρησιμοποιηθεί για την απλοποίηση της διαδικασίας επίλυσης, παρέχοντας σε μερικές περιπτώσεις εκθετική μείωση της πολυπλοκότητας.

Είναι αρκετά εύκολο να δούμε ότι σε ένα πρόβλημα ικανοποίησης περιορισμών μπορεί μπορεί να δοθεί μια αυξητική διατύπωση, όπως σε ένα συνηθισμένο πρόβλημα αναζήτησης, με το παρακάτω τρόπο:

Αρχική κατάσταση: Η κενή ανάθεση τιμών {}, όπου δεν έχει δοθεί τιμή σε καμία από τις μεταβλητές
Συνάρτηση διαδόχων: Μπορεί να δοθεί τιμή σε οποιαδήποτε μεταβλητή δεν έχει ήδη δοθεί, εφόσον αυτό δεν συγκρούεται με προηγούμενες αναθέσεις τιμών σε μεταβλητές
Έλεγχος στόχου: Η τρέχουσα ανάθεση τιμών είναι πλήρης
Κόστος διαδρομής: Ένα σταθερό κόστος για κάθε βήμα

Κάθε λύση πρέπει να είναι πλήρης ανάθεση τιμών, και επομένως εμφανίζεται σε βάθος n αν υπάρχουν n μεταβλητές. Επίσης, το δέντρο αναζήτησης εκτείνεται μόνο μέχρι βάθος n. Γι'αυτούς τους λόγους, οι αλγόριθμοι αναζήτησης πρώτα σε βάθος είναι δημοφιλείς στα προβλήματα ικανοποίησης περιορισμών. Συμβαίνει, επίσης, η διαδρομή μέσω της οποίας φτάνουμε σε μια λύση να είναι αδιάφορη. Συνεπώς, μπορούμε επίσης να χρησιμοποιήσουμε μια διατύπωση με πλήρεις καταστάσεις, στην οποία κάθε κατάσταση είναι πλήρης ανάθεση τιμών που μπορεί να ικανοποιεί ή να μην ικανοποιεί τους περιορισμούς. Οι μέθοδοι τοπικής αναζήτησης λειτουργούν καλά με μια τέτοια διατύπωση.

Το απλούστερο είδος προβλημάτων ικανοποίησης περιορισμών χρησιμοποιεί μεταβλητές οι οποίες είναι διακριτές και έχουν πεπερασμένα πεδία. Αν το μέγιστο πεδίο οποιασδήποτε μεταβλητής σε ένα πρόβλημα ικανοποίησης περιορισμών με n μεταβλητές έχει μέγεθος d, τότε ο αριθμός των δυνατών πλήρων αναθέσεων τιμών είναι $O(d^n)$, δηλαδή εκθετικό ως προς τον αριθμό των μεταβλητών. Σε αυτά τα προβλήματα ανήκουν και τα προβλήματα ικανοποίησης περιορισμών Boole, που οι μεταβλητές έχουν τιμή είτε ψευδής, είτε αληθής. Τα προβλήματα ικανοποίησης περιορισμών Boole περιλαμβάνουν ως ειδικές περιπτώσεις μερικά NP- προβλήματα, όπως τα 3SAT. Στην χειρότερη περίπτωση, λοιπόν δεν μπορούμε να αναμένουμε ότι τα προβλήματα ικανοποίησης περιορισμών με πεπερασμένα πεδία μπορούν να επιλύονται σε χρόνο μικρότερο του εκθετικού. Στις πρακτικές εφαρμογές, όμως, οι αλγόριθμοι γενικής χρήσης για τα προβλήματα ικανοποίησης περιορισμών μπορούν να επιλύουν προβλήματα πολλές τάξεις μεγέθους, μεγαλύτερα από εκείνα που μπορούν να επιλύονται με αλγορίθμους αναζήτησης γενικής χρήσης.

Οι διακριτές μεταβλητές μπορούν επίσης να έχουν άπειρα πεδία, για παράδειγμα το σύνολο των ακεραίων ή το σύνολο των συμβολοσειρών. Με

άπειρα πεδία δεν είναι δυνατόν να περιγράφονται οι περιορισμοί με απαρίθμηση όλων των επιτρεπτών συνδυασμών τιμών. Πρέπει να χρησιμοποιηθεί μια γλώσσα περιορισμών. Επίσης, δεν είναι δυνατόν να επιλύονται αυτοί οι περιορισμοί με απαρίθμηση όλων των δυνατών αναθέσεων, επειδή υπάρχουν άπειρες τέτοιες. Υπάρχουν ειδικοί αλγόριθμοι επίλυσης για γραμμικούς περιορισμούς σε ακέραιες μεταβλητές. Μπορεί να αποδειχτεί ότι δεν υπάρχει αλγόριθμος για την επίλυση γενικών μη γραμμικών περιορισμών σε ακέραιες μεταβλητές. Σε μερικές περιπτώσεις μπορούμε να ανάγουμε προβλήματα περιορισμών για ακέραιους σε προβλήματα με πεπερασμένα πεδία φράζοντας απλώς τις τιμές των άλλων μεταβλητών.

Τα προβλήματα ικανοποίησης περιορισμών με συνεχή πεδία είναι πολύ συνηθισμένα στον πραγματικό κόσμο και μελετώνται εκτεταμένα στο πεδίο της επιχειρησιακής έρευνας. Η γνωστότερη κατηγορία προβλημάτων ικανοποίησης περιορισμών είναι τα προβλήματα γραμμικού προγραμματισμού, όπου οι περιορισμοί πρέπει να είναι γραμμικές ανισότητες που σχηματίζουν μια κυρτή περιοχή. Τα προβλήματα γραμμικού προγραμματισμού μπορούν να επιλύονται σε χρόνο πολυωνυμικό ως προς τον αριθμό των μεταβλητών. Έχουν, επίσης, μελετηθεί προβλήματα με διαφορετικούς τύπους περιορισμών και αντικειμενικών συναρτήσεων, προβλήματα τετραγωνικού προγραμματισμού, κωνικού προγραμματισμού δεύτερης τάξης κλπ.

Εκτός από τους τύπους των μεταβλητών που μπορούν να εμφανίζονται στα προβλήματα ικανοποίησης περιορισμών, είναι χρήσιμο να εξετάσουμε τους τύπους των περιορισμών. Ο απλούστερος τύπος είναι ο μοναδιαίος περιορισμός, που περιορίζει την τιμή μίας μεταβλητής μόνο. Κάθε μοναδιαίος περιορισμός μπορεί να απαλειφτεί με απλή προ-επεξεργασία του πεδίου της αντίστοιχης μεταβλητής, ώστε να αφαιρεθεί οποιαδήποτε τιμή παραβιάζει τον περιορισμό. Ένας δυαδικός περιορισμός συσχετίζει δύο μεταβλητές. Δυαδικό πρόβλημα περιορισμών είναι αυτό που έχει μόνο δυαδικούς περιορισμούς.

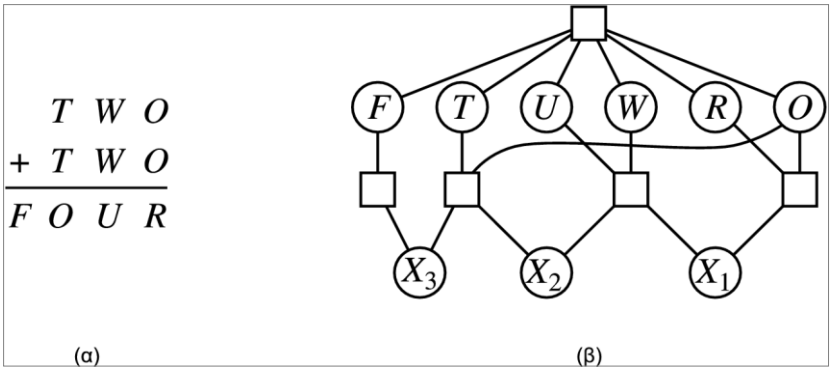
Οι περιορισμοί υψηλότερης τάξης περιλαμβάνουν τρεις ή περισσότερες μεταβλητές. Ένα γνωστό παράδειγμα είναι η αλφαριθμητικοί γρίφοι. Συνήθως απαιτείται κάθε γράμμα ενός κρυπταριθμητικού γρίφου να αντιπροσωπεύει ένα διαφορετικό ψηφίο. Στην περίπτωση της εικόνας 2α αυτό θα μπορούσε να αναπαρασταθεί με τον περιορισμό 6 μεταβλητών *Alldiff* (*F* , *T* , *U* , *W* , *R* , *O*). Εναλλακτικά, μπορεί να αναπαρασταθεί συλλογή δυαδικών περιορισμών όπως F ≠T. Οι περιορισμοί των προσθέσεων στις τέσσερις στήλες του γρίφου περιλαμβάνουν επίσης πολλές μεταβλητές που μπορεί να γραφτούν ως:

$O + O = R + 10 \times X1$

$X1 + W + W = U + 10 \times X2$

$X2 + T + T = O + 10 \times X3$

$X3 = F$



Εικόνα 4

όπου X1, X2, X3 είναι βοηθητικές μεταβλητές που αντιπροσωπεύουν το κρατούμενο ψηφίο (0 ή 1) που μεταφέρεται στην επόμενη στήλη. Οι περιορισμοί υψηλότερης τάξης μπορούν να αναπαρίστανται με ένα υπεργράφημα περιορισμών, όπως αυτό της εικόνας 2β. Προσέχουμε πως ο περιορισμός Alldif("όλα διαφορετικά") μπορεί να αναλυθεί σε δυαδικούς περιορισμούς F≠T κοκ. Μάλιστα, κάθε περιορισμός πεπερασμένων πεδίων υψηλότερης τάξης μπορεί να αναχθεί σε ένα σύνολο δυαδικών περιορισμών, αν εισαχθούν αρκετές βοηθητικές μεταβλητές. Στα προβλήματα που θα ασχοληθούμε παρακάτω, ο τύπος των περιορισμών είναι δυαδικός.

Οι παραπάνω περιορισμοί είναι όλοι απόλυτοι περιορισμοί, που η παραβίαση τους προκαλεί απόρριψη μιας ενδεχόμενης λύσης. Πολλά προβλήματα ικανοποίησης περιορισμών του πραγματικού κόσμου περιλαμβάνουν περιορισμούς προτίμησης που υποδεικνύουν ποιες λύσεις είναι προτιμότερες. Οι περιορισμοί προτίμησης μπορούν συχνά να κωδικοποιούνται ως κόστη στις αναθέσεις τιμών σε μεμονωμένες μεταβλητές. Με αυτή την διατύπωση, τα προβλήματα ικανοποίησης περιορισμών με προτιμήσεις μπορούν να επιλύονται με τη χρήση μεθόδων αναζήτησης με βελτιστοποίηση, είτε βασισμένων στη διαδρομή είτε τοπικών.

2.3 Τοπική Αναζήτηση στα Προβλήματα περιορισμών

Οι αλγόριθμοι τοπικής αναζήτησης αποδεικνύονται πολύ αποτελεσματικοί για την επίλυση πολλών προβλημάτων ικανοποίησης περιορισμών. Χρησιμοποιούν μια διατύπωση με πλήρεις καταστάσεις: Η αρχική κατάσταση αναθέτει τιμές σε κάθε μεταβλητή και η συνάρτηση διαδόχων συνήθως αναλαμβάνει να αλλάζει την τιμή μίας-μίας μεταβλητής.

Για παράδειγμα, στο πρόβλημα τον 8 βασιλισσών η αρχική κατάσταση θα μπορούσε να είναι μια τυχαία διάταξη 8 βασιλισσών σε 8 στήλες και η συνάρτηση διαδόχων παίρνει μια βασίλισσα και εξετάζει τις κινήσεις της σε άλλες θέσεις της στήλης. Μια άλλη δυνατότητα θα ήταν να ξεκινήσουμε με 8 βασίλισσες, μια ανά στήλη σε μια μετάθεση των 8 γραμμών και να παράγουμε μια διαδοχική κατάσταση με αντιμετάθεση των γραμμών δύο βασιλισσών. Έχουμε αναλύσει ήδη ένα παράδειγμα τοπικής αναζήτησης για την επίλυση

προβλημάτων ικανοποίησης περιορισμών: την εφαρμογή της αναρρίχησης λόφων στο πρόβλημα των n βασιλισσών. Ένα άλλο παράδειγμα είναι ο αλγόριθμος WALKSAT στην επίλυση προβλημάτων ικανοποιησιμότητας ο οποίος δεν θα μας απασχολήσει στην εργασία.

Άλλο πλεονέκτημα της τοπικής αναζήτησης είναι ότι μπορεί να χρησιμοποιηθεί σε online περιβάλλον όπου το πρόβλημα αλλάζει. Αυτό είναι ιδιαίτερα σημαντικό στα προβλήματα προγραμματισμού, όπου ο εβδομαδιαίος χρονοπρογραμματισμός αεροπορικών πτήσεων μπορεί να περιλαμβάνει χιλιάδες πτήσεις και δεκάδες χιλιάδες τοποθετήσεις προσωπικού, αλλά η κακοκαιρία σε ένα αεροδρόμιο μπορεί να κάνει το χρονοπρογραμματισμό ανέφικτο. Θα θέλαμε να επισκευάσουμε το χρονοπρογραμματισμό με ένα ελάχιστο αριθμό αλλαγών. Αυτό μπορεί να γίνει εύκολα με τον αλγόριθμο τοπικής αναζήτησης που ξεκινά από το τρέχοντα προγραμματισμό. Μια αναζήτηση με υπαναχώρηση με νέο σύνολο περιορισμών απαιτεί συνήθως πολύ περισσότερο χρόνο και θα μπορούσε να βρει λύση με πολλές αλλαγές σε σχέση με το τρέχοντα προγραμματισμό.

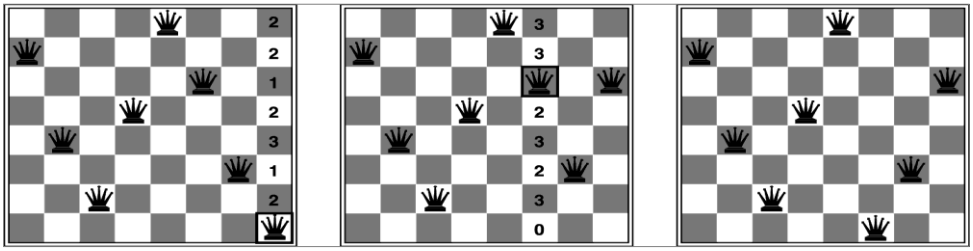
Για την επιλογή νέας τιμής για μια μεταβλητή ο πιο προφανής ευρετικός μηχανισμός είναι να επιλέγεται η τιμή που προκύπτει με ελάχιστο αριθμό συγκρούσεων με άλλες μεταβλητές, με τον ευριστικό μηχανισμό ελάχιστων συγκρούσεων (Min-conflicts). Όταν αναφερόμαστε σε **αριθμό (πλήθος) συγκρούσεων**, εννοούμε τον αριθμό των περιορισμών που παραβιάζει η τιμή μίας μεταβλητής σε σχέση με τις αναθέσεις τιμών στις υπόλοιπες μεταβλητές του προβλήματος. Ο αλγόριθμος παρουσιάζεται παρακάτω και η εφαρμογή περιγράφεται οπτικά στην εικόνα 5. Αυτόν τον αλγόριθμο θα υλοποιήσουμε στα παρακάτω κεφάλαια.

Αλγόριθμος Ελάχιστων Συγκρούσεων (Min-conflicts)

Η μέθοδος των ελάχιστων συγκρούσεων είναι εκπληκτικά αποτελεσματική για πολλά προβλήματα περιορισμών, ιδιαίτερα αν τους δοθεί μια λογική αρχική κατάσταση. Το εντυπωσιακό είναι ότι στο πρόβλημα των n βασιλισσών, αν δεν μετρήσετε την αρχική τοποθέτηση των βασιλισσών, ο χρόνος εκτέλεσης της μεθόδου των ελάχιστων συγκρούσεων είναι σχεδόν ανεξάρτητος από το μέθοδος του προβλήματος. Επιλύει ακόμα και το πρόβλημα ενός εκατομμυρίου βασιλισσών σε 50 βήματα κατά μέσο όρο. Αυτή η σημαντική παρατήρηση αποτέλεσε το ερέθισμα για πολύ μεγάλη ερευνητική προσπάθεια τη δεκαετία του 1990 πάνω στη τοπική αναζήτηση και τη διάκριση μεταξύ εύκολων και δύσκολων προβλημάτων. Σε γενικές γραμμές, το πρόβλημα n βασιλισσών είναι εύκολο για την τοπική αναζήτηση επειδή οι λύσεις είναι πυκνά κατανεμημένες σε όλο το χώρο καταστάσεων. Η μέθοδος Ελάχιστων Συγκρούσεων είναι επίσης κατάλληλη για δύσκολα προβλήματα. Για παράδειγμα, έχει χρησιμοποιηθεί για το χρονοπρογραμματισμό των παρατηρήσεων στο διαστημικό τηλεσκόπιο Hubble, μειώνοντας το χρόνο που χρειάζεται για τον χρονοπρογραμματισμό μιας εβδομάδας παρατηρήσεων από τρεις βδομάδες σε 10 λεπτά περίπου.

Όπως βλέπουμε στον αλγόριθμο παρακάτω, ο Min–conflicts αλγόριθμος ξεκινάει με μια αρχική κατάσταση. Στην συνέχεια διαλέγει τυχαία μία μεταβλητή που παραβιάζει ένα περιορισμό. Βρίσκει την τιμή που ελαχιστοποιεί τις συγκρούσεις της μεταβλητής και την θέτει ως τη τιμή της επιλεγμένης μεταβλητής στην τρέχουσα κατασταση του αλγορίθμου. Αυτή η διαδικασία επαναλαμβάνεται max_steps φορές.

```
function MIN-CONFLICTS(csp, max_steps) returns a solution or failure
  inputs: csp, a constraint satisfaction problem
        max_steps, the number of steps allowed before giving up
  current← an initial assignment for csp
  for i=1 to max_steps do
     if current is a solution of csp then return current
          var← a randomly chosen, conflicted variable from VARIABLES[csp]
          value← the value v for var that minimizes
CONFLICTS(var,v,current,csp)
          set var = value in current
 return failure
```



Εικόνα 5

Αλγόριθμος Ακτινικής Τοπικής Αναζήτησης(Local Beam Search)

Ο άλλος αλγόριθμος που θα υλοποιήσουμε στην διπλωματική είναι ο αλγόριθμος της Ακτινικής Τοπικής Αναζήτησης, ο οποίος παρακολουθεί *k* καταστάσεις αντί μόνο μία. Ξεκινά με *k* τυχαία παραγόμενες καταστάσεις. Σε κάθε βήμα παράγονται όλοι οι διάδοχοι και των *k* καταστάσεων. Αν οποιαδήποτε από αυτές ικανοποιεί τον στόχο ,ο αλγόριθμος σταματά. Αλλιώς, επιλέγει τους *k* καλύτερους διαδόχους από όλη τη λίστα και επαναλαμβάνει. Ο αλγόριθμος παρουσιάζεται παρακάτω.

Function **LOCAL BEAM SEARCH**(csp,max_steps,κ) returns **a solution** or **failure**

inputs: csp, a constraint satisfaction problem with n variables

max_steps, the number of steps allowed before giving up

k: number of randomly generated states
current_states: an array of k complete assignments to variables

best_neighbors: an array of k*k complete assignments to variables

for i=1 **to** k **do**
current_states[i] ← an initial random assignment to all n variables
endfor

if any state in current_states is a solution return it

for i=1 **to** max_steps **do**

for j=1 **to** k **do**
best_neighbors[j] ← the k neighbor states of current_states[j] that minimize conflicts

if the best state in best_neighbors[j] is a solution return it

endfor

current_states ← the k best states in best_neighbors

endfor

return failure

3.ΥΛΟΠΟΙΗΣΗ ΑΛΓΟΡΙΘΜΩΝ

Στην εργασία υλοποιούμε τον αλγόριθμο Ελάχιστων Συγκρούσεων και τον αλγόριθμο Ακτινικής Τοπικής Αναζήτησης σε προβλήματα ικανοποίησης περιορισμών. Για κάθε πρόβλημα μας δίνονται οι εξής πληροφορίες: ο αριθμός των μεταβλητών του προβλήματος, το πεδίο τιμών της κάθε μεταβλητής (domain) και οι περιορισμοί που συμμετέχει κάθε μεταβλητή. Το πρόβλημα, πιο αναλυτικά, έχει ένα αριθμό domains που αντιστοιχίζονται σε κάθε μεταβλητή. Για παράδειγμα, ένα πρόβλημα μπορεί να έχει 2 domains και όλες οι μεταβλητές να έχουν ως πεδίο τιμών ένα από αυτά τα δύο domains. Επίσης, κάθε περιορισμός δίνει τις επιτρεπόμενες τιμές των μεταβλητών που συμετέχουν σε αυτόν. Οι μεταβλητές που συμμετέχουν σε κάθε περιορισμό, στα προβλήματα που θα ασχοληθούμε, είναι πάντα 2.

3.1 Κύριο πρόγραμμα

Το πρόγραμμα υλοποιήθηκε σε γλώσσα C++. Η αναπαράσταση του προβλήματος γίνεται με δομές πινάκων. Παρακάτω θα αναλύσουμε τις κυριότερες από αυτές.

3.1.1 ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

Για την αποθήκευση των πληροφοριών των προβλημάτων περιορισμών δημιουργήσαμε ένα header με συναρτήσεις.

Οι βασικές δομές τους είναι οι εξής:

Στη συνάρτηση var::fill ο μονοδιάστατος πίνακας **var [n]** που αποθηκεύει για κάθε μεταβλητή, κάθε προβλήματος τον αριθμό του domain που της αντιστοιχεί. Για παράδειγμα, το var[1]=1 περιέχει την τιμή του domain που αντιστοιχεί στην μεταβλητή x_1, στην συγκεκριμένη περίπτωση είναι το 1.

Στη συνάρτηση dom::fill ο δισδιάστατος πίνακας **dom[N][N]** που αποθηκεύει τις τιμές που αντιστοιχούν σε κάθε πεδίο τιμών (domain). Σε συνδυασμό με το προηγούμενο παράδειγμα, το dom[1][N] περιέχει τις τιμές που ανήκουν στο πεδίο τιμών 1, οι οποίες έχουν πλήθος N. Άρα η μεταβλητή x_1 έχει πεδίο ορισμού τις N τιμές που είναι αποθηκευμένες στη πρώτη γραμμή του πίνακα dom.

Στη συνάρτηση con::fill για πρακτικούς λόγους δημιουργούμε τον δισδιάστατο πίνακα **con[N][N]** που αποθηκεύει τους επιτρεπόμενους συνδυασμούς τιμών για τις μεταβλητές που αποθηκεύονται στις 2 πρώτες θέσεις του πίνακα. Για παράδειγμα, το con[0][0]=0 περιέχει τη πρώτη μεταβλητή που συμμετέχει στους παρακάτω περιορισμούς, το x_0, το con[0][1]=1 την δεύτερη, το x_1. Στη συνέχεια, το con[0][2]=2 είναι η επιτρεπόμενη τιμή για την πρώτη μεταβλητή, στη συγκεκριμένη περίπτωση τη μεταβλητή x_0 και το con[0][3]=3 είναι η επιτρεπόμενη τιμή για την μεταβλητή x_1. Άρα, ο πρώτος επιτρεπόμενος συνδυασμός για τις μεταβλητές x_0 και x_1 είναι το (2,3) και αυτό συνεχίζεται για

όλους τους επιτρεπόμενους συνδυασμούς των 2 μεταβλητών αυτών. Για τον επόμενο συνδυασμό μεταβλητών θα έχουμε con[1][0] και con[1][1] και κατ'αυτόν τον τρόπο συνεχίζεται η ίδια διαδικασία.

Έχουμε τον τετραδιάστατο πίνακα **con[n][n][N][N]** ο οποίος έχει μέγεθος n(αριθμός μεταβλητών)*n(αριθμός μεταβλητών)*N(αριθμός της μεγαλύτερης επιτρεπόμενης τιμής μεταβλητών)*N (αριθμός της μεγαλύτερης επιτρεπόμενης τιμής μεταβλητών). Αποθηκεύει τα επιτρεπόμενα ζεύγη τιμών, βάζοντας ένα 1 στο κελί του πίνακα ενός επιτρεπόμενου συνδυασμού τιμών. Για παράδειγμα, αν ένας επιτρεπόμενος συνδυασμός είναι, για τις μεταβλητές x_0 και x_4 , το (0,0) τότε το κελί con[0][4][0][0] θα είναι 1. Τα υπόλοιπα κελιά που δεν περιέχουν επιτρεπόμενους συνδυασμούς παίρνουν την τιμή 0.

3.1.2. ΛΕΙΤΟΥΡΓΙΑ ΠΡΟΓΡΑΜΜΑΤΟΣ

Τα κοινά σημεία του προγράμματος και για τους δύο αλγορίθμους αφορούν την αποθήκευση των παραμέτρων των προβλημάτων ικανοποίησης περιορισμών δηλαδή των μεταβλητών, των domains των μεταβλητών και των περιορισμών με τις δομές που αναλύσαμε παραπάνω.

3.2 Αλγόριθμος Ελάχιστων Συγκρούσεων

3.2.1 ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

Οι βασικές δομές δεδομένων σε αυτόν τον αλγόριθμο είναι οι εξής:

randomval[i]: μονοδιάστατος πίνακας που αποθηκεύονται οι τυχαίες τιμές για κάθε μεταβλητή i που παράγονται με την συνάρτηση rand()(συνάρτηση της C++).

conflict[i]: μονοδιάστατος πίνακας στον οποίο αποθηκεύεται το πλήθος των συγκρούσεων περιορισμών για κάθε μεταβλητή xi με την τιμή randomval[i].

varconflicts[z]: μονοδιάστατος πίνακας στον οποίο αποθηκεύεται το πλήθος των συγκρούσεων για κάθε τιμή z, της μεταβλητής που επιλέχτηκε τυχαία από τον πίνακα randomval και δεν έχει μηδενικές συγκρούσεις, με βάση την συνάρτηση rand().

minconflicts: αποθηκεύει την ελάχιστη τιμή του πίνακα varconflicts.

bests[i]: μονοδιάστατος πίνακας στον οποίο αποθηκεύονται οι καλύτερες αναθέσεις τιμών για κάθε μεταβλητή, δηλαδή η ανάθεση τιμής σε κάθε μεταβλητή που προκαλεί τις λιγότερες συγκρούσεις περιορισμών.

3.2.2. ΛΕΙΤΟΥΡΓΙΑ ΑΛΓΟΡΙΘΜΟΥ ΕΛΑΧΙΣΤΩΝ ΣΥΓΚΡΟΥΣΕΩΝ

Ο αλγόριθμος αρχικά δίνει τυχαίες τιμές σε όλες τις μεταβλητές του προβλήματος, όπως βλέπουμε στην γραμμή 3 του αλγόριθμου που παρουσιάζεται παρακάτω. Στη συνέχεια, υπολογίζει το πλήθος των συγκρούσεων όλων των μεταβλητών, ελέγχοντας πόσους περιορισμούς παραβιάζει κάθε μεταβλητή με τη τιμή που της ανατέθηκε (γραμμή 6). Ελέγχει αν αυτή η ανάθεση τιμών είναι λύση (μηδενικές συγκρούσεις), όπως βλέπουμε στη γραμμή 7 του αλγορίθμου. Αν είναι τότε ο αλγόριθμος σταματάει επιστρέφοντας τη λύση (γραμμή 8). Αν όμως δεν είναι λύση, επιλέγει τυχαία μία μεταβλητή της οποίας η ανάθεση παραβιάζει κάποιο περιορισμό (γραμμή 10). Υπολογίζει το πλήθος των συγκρούσεων για κάθε ανάθεση τιμής στη μεταβλητή που έχει επιλέξει και διαλέγει την τιμή που προκαλεί τις ελάχιστες συγκρούσεις (γραμμή 11-12). Στη γραμμή 13 θέτει ως τρέχουσα κατάσταση (randomval) τη πλήρη ανάθεση τιμών που παραβιάζει τους λιγότερους περιορισμούς(minconflict). Ξαναεπιστρέφει στον πρώτο υπολογισμό του πλήθους των συγκρούσεων και τον έλεγχο αν είναι λύση (γραμμή 6-7) και επαναλαμβάνει όσες φορές επιλέξει ο χρήστης(number1), αυτή τη διαδικασία. Όλη η διαδικασία του αλγορίθμου επαναλαμβάνεται, επίσης όσες φορές επιλέξει ο χρήστης(number), ουσιαστικά επανεκκινόντας το πρόγραμμα, δηλαδή επαναλαμβάνοντας των αλγόριθμο με νέες τυχαίες αρχικές καταστάσεις.

3.2.3. ΥΛΟΠΟΙΗΣΗ ΑΛΓΟΡΙΘΜΟΥ ΕΛΑΧΙΣΤΩΝ ΣΥΓΚΡΟΥΣΕΩΝ

Input: constraint satisfaction problem , number, number1

Output: Either a solution, or the best assignment.

```
1.  restarts=0;
2.  while restarts < number
3.      randomval ← give randomly chosen values to all the variables
4.      repeat=0;
5.      while repeat < number1
6.          conflicts← calculate the conflicts of the constraints for randomval
    assignment
7.       If  randomval  is a solution (conflicts=0)
8.           return randomval
9.       else
10.          var← a randomly chosen, conflicted variable
11.          varconflicts← calculate the conflicts of constraints for every
    permitted value of var
12.          minconflict ← choose the minimum of  varconflicts array
```

13. **randomval**← assignment of variables with the minconflict
14. **endif**
15. **endwhile**
16. **endwhile**

3.2.4 ΠΑΡΑΔΕΙΓΜΑ ΑΛΓΟΡΙΘΜΟΥ

Έστω ένα πρόβλημα περιορισμών με μεταβλητές a,b,c,d με πεδίο τιμών D={0,1} και περιορισμούς $a \neq b$, $b \neq c$, $c \neq d$.

Βήμα 1^{ο} : Ανάθεση τυχαίων τιμών σε όλες τις μεταβλητές:

a=0 b=0 c=1 d=0

Βήμα 2^{ο} : Υπολογισμός συγκρούσεων κάθε μεταβλητής για κάθε επιτρεπόμενη τιμή.

Για a=0: 1 σύγκρουση: a=b

Για b=0: 1 σύγκρουση: a=b

Για c=1: 0 συγκρούσεις

Για d=0: 0 συγκρούσεις

Βήμα 3^{ο} : Τυχαία επιλογή μιας μεταβλητής που συμμετέχει σε σύγκρουση περιορισμών:

Έστω ότι επιλέγει την μεταβλητή a

Βήμα 4^{ο} : Υπολογισμός συγκρούσεων της επιλεγμένης μεταβλητής για κάθε επιτρεπόμενης τιμή και επιλογή της τιμής με τις λιγότερες συγκρούσεις.

Για a=0:1 σύγκρουση: a=b

Για a=1:0 συγκρούσεις

Επιλογή της τιμής 1. Έλεγχος αν είναι λύση. Είναι λύση.

Εάν δεν ήταν:

Παίρνουμε τη κατάσταση της επιλεγόμενης τιμής ως την αρχική ανάθεση τιμών και επαναλαμβάνουμε τη διαδικασία από το Βήμα 2 για **number1** φορές.

3.3 Αλγόριθμος Ακτινικής Τοπικής Αναζήτησης

3.3.1 ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

Οι βασικές δομές δεδομένων σε αυτόν τον αλγόριθμο είναι οι εξής:

randomval[i][j]: δισδιάστατος πίνακας οπου αποθηκεύονται οι k τυχαίες τιμές για κάθε μεταβλητή i που παράγονται με την συνάρτηση rand().

conflict[i][j][l]: τρισδιάστατος πίνακας στον οποίο αποθηκεύεται το πλήθος περιορισμών που παραβιάζονται για κάθε επιτρεπόμενη τιμή l κάθε μεταβλητής i, και για τις k αρχικές καταστάσεις(randomval)**.**

best[j][qu]: δισδιάστατος πίνακας με τις k μικρότερες τιμές του πλήθους των συγκρούσεων (conflict[i][j][l]) για κάθε j τυχαία ανάθεση τιμών.

bestk[j][qu][i]: τρισδιάστατος πίνακας με τις k καλύτερες πλήρεις αναθέσεις τιμών (για όλες τις μεταβλητές i) για κάθε j, οι οποίες αντιστοιχούν στο πλήθος των συγκρούσεων best[j][qu]

best1[qu]: μονοδιάστατος πίνακας με τις k μικρότερες τιμές του best[j][qu].

bestk1[qu][i]: δισδιάστατος πίνακας με τις k καλύτερες πλήρης αναθέσεις τιμών, οι οποίες αντιστοιχούν στο πλήθος των συγκρούσεων best1[qu].

3.3.2. ΛΕΙΤΟΥΡΓΙΑ ΑΛΓΟΡΙΘΜΟΥ ΑΚΤΙΝΙΚΗΣ ΤΟΠΙΚΗΣ ΑΝΑΖΗΤΗΣΗΣ

Ο αλγόριθμος αρχικά δημιουργεί τυχαία k αρχικές καταστάσεις, ο αριθμός k επιλέγεται από το χρήστη, δηλαδή k σύνολα αναθέσεων τιμών στις μεταβλητές του προβλήματος, όπως βλέπουμε στις γραμμές 1-3 στον ψευδοκώδικα που ακολουθεί. Στη συνέχεια, για κάθε μια από τις k καταστάσεις και κάθε μια από τις μεταβλητές υπολογίζει το πλήθος των συγκρούσεων για όλες τις επιτρεπόμενες τιμές της (γραμμή 7). Ταξινομεί τον πίνακα συγκρούσεων κατά αύξουσα σειρά (γραμμή 9) και επιλέγει τις k πρώτες (μικρότερες) τιμές του πίνακα για κάθε μία από τις k τυχαίες αναθέσεις και τις αποθηκεύει στον πίνακα best[j][qu] (γραμμή 10). Επίσης, αποθηκεύει τις αντίστοιχες αναθέσεις τιμών στον πίνακα bestk[j][j][i] (γραμμή 13). Ελέγχει αν κάποια από τις αναθέσεις τιμών είναι λύση (μηδενικές συγκρούσεις), όπως βλέπουμε στη γραμμή 12 του αλγορίθμου. Αν είναι, τότε ο αλγόριθμος σταματάει επιστρέφοντας τη λύση(γραμμή 15). Αν δεν είναι λύση τότε ταξινομεί κατά αύξουσα σειρά τον πίνακα best[j][qu] (γραμμή 17) και επιλέγει τις k πρώτες (μικρότερες) τιμές του πίνακα (γραμμή 16). Επίσης, αποθηκεύει τις αντίστοιχες αναθέσεις τιμών στο πίνακα bestk1[j] (γραμμή 17).

Ελέγχει αν κάποια από τις ανάθεση τιμών είναι λύση(μηδενικές συγκρούσεις), όπως βλέπουμε στην γραμμή 20 και 21 του αλγορίθμου. Αν δεν είναι λύση θέτει ως k αρχικές καταστάσεις (randomval[j]) τις πλήρεις αναθέσεις τιμών που αντιστοιχούν στο πλήθος συγκρούσεων best1[j], δηλαδή τις bestk1[j] (γραμμή 25). Ο αλγόριθμος επαναλαμβάνεται από τη γραμμή 6 όσες φορές επιλέξει ο χρήστης(max_steps).

3.3.3. ΥΛΟΠΟΙΗΣΗ ΑΛΓΟΡΙΘΜΟΥ ΑΚΤΙΝΙΚΗΣ ΤΟΠΙΚΗΣ ΑΝΑΖΗΤΗΣΗΣ

Input: constraint satisfaction problem, max_steps

Output: Either a solution, or the best assignment.

1. **for** 0< j<k
2. **randomval[j]** ← give randomly chosen values to all of the variables
3. **endfor**
4. i=0;
5. **while** i < max_steps
6. **for** 0< j <k
7. **conflicts[i][j][I]** ← calculate the conflicts of the constraints of all variables i by checking all their permitted values I for every j randomval assignment
8. **for** 0< qu < k
9. **sorted[j][qu]** ← sorted conflicts[i][j][I] from the minimum value to the maximum
10. **best[j][qu]** ← the k first values of sorted array for every j
11. **endfor**
12. **endfor**
13. **bestk[j][qu][i]** ← the assignment of best[j][j]
14. **if** any of the **bestk[j][qu][i]** is **a solution**
15. **return** solution
16. **else**
17. **sorted1[j][qu]** ← sorted best from the minimum value to the maximum
18. **best1[j]** ← the k first values of sorted1 array
19. **bestk1[j]** ← the k assignments of best1[j]
20. **if bestk1[j]** is **a solution**
21. **return** solution

```
22.          endif
23.        endif
24.       for   0< j<k
25.            randomval[j] ← bestk1[j]
26.       endfor
27.       i++
28.      endwhile
```

3.3.4 ΠΑΡΑΔΕΙΓΜΑ ΑΛΓΟΡΙΘΜΟΥ

Έστω ένα πρόβλημα περιορισμών με μεταβλητές a,b,c,d με πεδίο τιμών D={0,1} και περιορισμούς $a \neq b$, $b \neq c$, $c \neq d$. Θέτουμε κ=2.

Βήμα 1^{ο} : Ανάθεση κ=2 τυχαίων τιμών σε όλες τις μεταβλητές:

Για κ=0 : a=0 b=0 c=0 d=0

Για k=1 : a=0 b=1 c=1 d=0

Βήμα 2^{ο} : Υπολογισμός συγκρούσεων κάθε μεταβλητής για κάθε επιτρεπόμενη τιμή και για τις κ=2 καταστάσεις.

Για k=0 :

Για την μεταβλητή a:

a=0 3 συγκρούσεις: a=b, b=c, c=d
a=1 2 συγκρούσεις: b=c, c=d

Για την μεταβλητή b:

b=0 3 συγκρούσεις: a=b, b=c, c=d
b=1 1 σύγκρουση: c=d

Για την μεταβλητή c:

c=0 3 συγκρούσεις: a=b, b=c, c=d
c=1 1 σύγκρουση: a=b

Για την μεταβλητή d:

d=0 3 συγκρούσεις: a=b, b=c, c=d
d=1 2 συγκρούσεις: a=b, b=c

Για k=1 : Για την μεταβλητή a:

a=0 1 σύγκρουση: b=c
a=1 2 συγκρούσεις: a=b, b=c

Για την μεταβλητή b:

b=0 1 σύγκρουση: a=b
b=1 1 σύγκρουση: b=c

Για την μεταβλητή c:

c=0 1 σύγκρουση: c=d
c=1 1 σύγκρουση: b=c

Για την μεταβλητή d:

d=0 1 σύγκρουση: b=c
d=1 2 συγκρούσεις: b=c, c=d

Βήμα 3^{ο} : Ταξινόμηση και επιλογή των κ=2 καλύτερων περιπτώσεων(λιγότερες συγκρούσεις) και για τις 2 καταστάσεις:

Για k=0 : 2 καλύτερες περιπτώσεις:
b=1 1 σύγκρουση: c=d
c=1 1 σύγκρουση: a=b

Για k=1 : 2 καλύτερες περιπτώσεις:
a=0 1 σύγκρουση: b=c
d=0 1 σύγκρουση: b=c

Ελέγχουμε αν κάποιο είναι λύση, στο παράδειγμά μας κανένα δεν είναι λύση άρα:

Βήμα 4^{ο} : Ταξινόμηση και επιλογή των κ=2 καλύτερων περιπτώσεων (λιγότερες συγκρούσεις):

2 καλύτερες περιπτώσεις:

a=0 b=1 c=0 d=0

a=0 b=1 c=1 d=0

Παίρνουμε αυτές τις 2 περιπτώσεις ως την αρχική ανάθεση τιμών και επαναλαμβάνουμε την διαδικασία από το Βήμα 2 για **max_step** φορές.

4.ΑΠΟΤΕΛΕΣΜΑΤΑ

Οι αλγόριθμοι που αναλύσαμε και υλοποιήσαμε στο προηγούμενο κεφάλαιο εκτελέστηκαν επανειλημμένα για την αξιολόγησή τους. Τους εκτελέσαμε 9 φορές για κάθε ανάθεση τιμής, στις παραμέτρους που επιλέγονται από τον χρήστη, όπως και για κάθε πρόβλημα. Τα προβλήματα επιλέχτηκαν ώστε να έχουν διαφορετικό μέγεθος και επίπεδο δυσκολίας για να έχουμε πιο αντιπροσωπευτικά δείγματα. Συγκεκριμένα τα προβλήματα είναι τα εξής:

- **langford-2-4-ext :** Πρόβλημα 8 μεταβλητών, οι οποίες αντιστοιχούν σε 1 domain. Οι περιορισμοί του προβλήματος είναι 32. Αποτελεί το πιο εύκολο από τα προβλήματα που έχουμε επιλέξει.

- **hanoi-5_ext :** Πρόβλημα 30 μεταβλητών, οι οποίες αντιστοιχούν σε 3 domains. Οι περιορισμοί του προβλήματος είναι 29.

- **geo50-20-d4-75-1_ext :** Πρόβλημα 50 μεταβλητών, οι οποίες αντιστοιχούν σε 1 domain. Οι περιορισμοί του προβλήματος είναι 472.

- **composed-75-1-80-6_ext :** Πρόβλημα 83 μεταβλητών, οι οποίες αντιστοιχούν σε 1 domain. Οι περιορισμοί του προβλήματος είναι 702.

- **BlackHole-4-7-h-9_ext :** Πρόβλημα 112 μεταβλητών, οι οποίες αντιστοιχούν σε 4 domains. Οι περιορισμοί του προβλήματος είναι 1262.

- **qwh-15-106-1_ext :** Πρόβλημα 225 μεταβλητών, οι οποίες αντιστοιχούν σε 16 domains. Οι περιορισμοί του προβλήματος είναι 3150.

- **driverlogw-08c-sat_ext :** Το μεγαλύτερο από τα προβλήματα που ασχοληθήκαμε. Πρόβλημα 408 μεταβλητών, οι οποίες αντιστοιχούν σε 8 domains. Οι περιορισμοί του προβλήματος είναι 9321.

4.1.Αποτελέσματα αλγορίθμου Ελάχιστων Συγκρούσεων

Οι παράμετροι που επιλέχτηκαν απο τον χρήστη, όπως είδαμε στο παραπάνω κεφάλαιο, για τον αλγόριθμο Ελάχιστων Συγκρούσεων είναι το number που αντιστοιχεί στις επανεκκινήσεις του αλγορίθμου(restarts) και το number1 που αντιστοιχεί στις εσωτερικές επαναλήψεις του αλγορίθμου (repeat).

Όπως θα δούμε παρακάτω, για τα προβλήματα που επιλέξαμε και με τη χρήση του αλγόριθμου Ελάχιστων Συγκρούσεων δεν βρέθηκε λύση για κανένα πρόβλημα. Οι πίνακες μας δείχνουν τον μέσο όρο του πλήθους των συγκρούσεων για κάθε πρόβλημα και για κάθε συνδυασμό τιμών.

langford-2-4-ext :

repeat / restarts	10	100	1000
10	43,11111111	36,77777778	36
100	36	36	36
1000	36	36	36

Παρατηρούμε ότι στο πρόβλημα μας αυτό, εκτός από την πρώτη εκτέλεση του για τιμές restarts=10 και repeat=10, οι συγκρούσεις παρ'όλο της διαφοράς μεγέθους των τιμών παραμένουν σταθερές. Άρα καταλήγουμε στο συμπέρασμα ότι ο αλγόριθμος έχει παγιδευτεί σε τοπικό βέλτιστο, από το οποίο είναι δύσκολο να φύγει.

hanoi-5_ext :

repeat / restarts	10	100	1000
10	891	880,4444444	877,6666667
100	889,3333333	888,7777778	876,7777778
1000	888,7777778	878,2222222	875,8888889

geo50-20-d4-75-1_ext :

repeat / restarts	10	100	1000
10	2364,444444	2360,111111	2358,111111
100	2356,111111	2354	2351,888889
1000	2348,444444	2346,777778	2342,333333

composed-75-1-80-6_ext :

repeat / restarts	10	100	1000
10	6761,777778	6726,888889	6708,666667
100	6756,444444	6720,333333	6703,333333
1000	6745,444444	6716	6694,333333

qwh-15-106-1_ext :

repeat / restarts	10	100	1000
10	50452,44444	50443,66667	50198,33333
100	50442,66667	50426,77778	50156,22222
1000	50430,22222	50402,44444	50093,77778

BlackHole-4-7-h-9_ext :

repeat / restarts	10	100	1000
10	12373,33333	12318,33333	12123,44444
100	12347,11111	12281,44444	12058,11111
1000	12322,22222	12236,44444	12008,33333

driverlogw-08c-sat_ext :

repeat / restarts	10	100	1000
10	164187,5556	163886,8889	162984,1111
100	164080	163822	162850,5556
1000	163955,1111	163610,1111	162782,4

Παρατηρώντας τους παραπάνω πίνακες βλέπουμε ότι όλα τα προβλήματα έχουν μια σταδιακή βελτίωση των αποτελεσμάτων τους κατά την αύξηση των επαναλήψεων και των επανεκκινήσεων, άρα όλα παρουσιάζουν τον ελάχιστο μέσο όρο συγκρούσεων τους για restarts=1000 και repeat=1000.(εξαίρεση αποτελεί το πρόβλημα langford-2-4-ext). Παρόλα αυτά φαίνεται ότι ο αλγόριθμος Ελάχιστων Συγκρούσεων δεν είναι καλή επιλογή για την επίλυση αυτών των προβλημάτων, καθώς στο τέλος αυτής της εκτέλεσης του συνεχίζουν να υπάρχουν πολλές συγκρούσεις. Αυτό πιθανών οφείλεται στη τοπολογία αυτών των προβλημάτων, τα οποία παρουσιάζουν πολλά βέλτιστα.

4.2.Αποτελέσματα αλγορίθμου Ακτινικής Αναζήτησης

Οι παράμετροι που επιλέχτηκαν από το χρήστη, όπως είδαμε στο παραπάνω κεφάλαιο, για τον αλγόριθμο Ακτινικής Αναζήτησης είναι το k που αντιστοιχεί στον αριθμό των αρχικών τυχαίων επανεκκινήσεων, στον αριθμό των γειτονικών καταστάσεων που παράγονται και τέλος τον αριθμό των επιλεγμένων καλύτερων τιμών και για κάθε παραγόμενη κατάσταση αλλά και για την τελική επιλογή και το max_steps που αντιστοιχεί στις εσωτερικές επαναλήψεις του αλγορίθμου (repeat).

Όπως θα δούμε παρακάτω, για τα προβλήματα που επιλέξαμε και με την χρήση του αλγόριθμου Ακτινικής Αναζήτησης δεν βρέθηκε λύση για κανένα πρόβλημα. Οι πίνακες μας δείχνουν τον μέσο όρο του πλήθους των συγκρούσεων για κάθε πρόβλημα και για κάθε συνδυασμό τιμών.

langford-2-4-ext :

k / repeat	2	10	20	49
10	36	36	36	36
100	36	36	36	36
1000	36	36	36	36

Παρατηρούμε ότι και ο αλγόριθμος Ακτινικής Αναζήτησης, στο πρόβλημα langford-2-4-ext, παρουσιάζει τον ίδιο μέσο όρο συγκρούσεων (δηλαδή πολλαπλές καταστάσεις με τον ίδιο αριθμό συγκρούσεων) για όλους τους συνδυασμούς τιμών, ακόμα και με τις μικρότερες, δηλαδή k = 2 και repeat=10. Αυτό μπορεί να σημαίνει ότι έχουμε κολλήσει σε ένα τοπικό βέλτιστο ή ίσως και σε ένα ολικό βέλτιστο, δηλαδή τη καλύτερη δυνατή κατάσταση.

hanoi-5_ext :

k / repeat	2	10	20	50
10	886,4444444	885,8888889	885,3333333	883,7777778
100	875,1111111	876,1111111	876,7777778	875,6666667
1000	875	876,2222222	876	875

geo50-20-d4-75-1_ext :

k / repeat	2	10	20	50
10	2305,777778	2301,888889	2302,222222	2295,333333
100	2243,444444	2247,777778	2244,555556	2239,4
1000	2245,666667	2250,4	2249	2240,222222

composed-75-1-80-6_ext :

k / repeat	2	10	20	50
10	6728,777778	6717,777778	6714,888889	6712,111111
100	6631,666667	6625,111111	6623,8	6621
1000	6631,444444	6624,8	6623,11111	6620,333333

BlackHole-4-7-h-9_ext :

k / repeat	2	10	20	50
10	12103,77778	12085,66667	12074,55556	12036,66667
100	11386,77778	11384,55556	11383,8	11381,4
1000	11379,55556	11376,55556	11374, 77778	11371, 55556

qwh-15-106-1_ext :

k / repeat	2	10	20	50
10	50403,11111	50394,66667	50391,22222	50383,66667
100	49888,55556	49915,11111	49912,66667	49908,66667
1000	49927,55556	49920, 66667	49910, 11111	49906, 11111

Παρατηρούμε, ότι η αύξηση των επαναλήψεων από 100 σε 1000 δεν επηρεάζει τα μεσαία και μικρά προβλήματα. Το πλήθος των συγκρούσεων παραμένει σχεδόν σταθερό για κάθε κατάσταση που καταλήξαμε, αφού οι τιμές διαφέρουν ελάχιστα, κάτι το οποίο οφείλεται κυρίως στην παράμετρο της τυχαιότητας. Δηλαδή στο γεγονός ότι πέρνουμε τυχαίες αρχικές τιμές.

driverlogw-08c-sat_ext :

k / repeat	2	10	20	50
10	164021,3333	163851,3333	163795,3333	163764,3333
100	162142,8889	162062,1111	162060,2222	162014,8889
1000	161454,8889	161448, 8889	161446,2222	161380,8889

Σε αντίθεση με τα υπόλοιπα προβλήματα, το πρόβλημα driverlogw-08c-sat_ext έχει μεγάλη βελτίωση του αριθμού ελάχιστων συγκρούσεων κατά την αύξηση του repeat από 100 σε 1000. Άρα μια τόσο μεγάλη αύξηση των επαναλήψεων του αλγορίθμου Ακτινικής Αναζήτησης επηρεάζει μόνο τα προβλήματα μεγάλου μεγέθους.

5.ΣΥΜΠΕΡΑΣΜΑΤΑ

Κατά την υλοποίηση των αλγορίθμων και την εκτέλεσή τους, όπως και κατά την εξέταση των αποτελεσμάτων των αλγορίθμων, κατέληξα σε κάποια συμπεράσματα.

Το πρώτο συμπέρασμα παρατηρήθηκε κατά την εκτέλεση των αλγορίθμων. Ο αλγόριθμος της Τοπικής Ακτινικής Αναζήτησης είναι πολύ πιο χρονοβόρος από τον αλγόριθμο Ελάχιστων Συγκρούσεων. Για να καταλάβουμε τη διαφορά στο χρόνο εκτέλεσης των δύο αλγορίθμων, θα πάρουμε τις περιπτώσεις των μικρότερων τιμών που έχουμε δώσει στις μεταβλητές για να τους συγκρίνουμε. Δηλαδή, για τον αλγόριθμο Ελάχιστων Συγκρούσεων θα πάρουμε την περίπτωση για restarts=10 και repeat=10 και για τον αλγόριθμο Τοπικής Ακτινικής Αναζήτησης για k=2 και repeat=10. Στο πρώτο και μικρότερο πρόβλημα (langford-2-4-ext), οι δύο αλγόριθμοι είναι αντίστοιχα γρήγοροι λόγω του μεγέθους και της ευκολίας του προβλήματος. Ενδεικτικό είναι ότι οι αλγόριθμοι εκτελέστηκαν σε 0 sec. Στα μεγαλύτερα προβλήματα όμως παρατηρούμε μεγάλες διαφορές.

Προβλήματα	Αλγόριθμός Ελάχιστων Συγκρούσεων	Αλγόριθμος Ακτινικής Αναζήτησης
hanoi-5_ext	1 sec	11 sec
geo50-20-d4-75-1_ext	1 sec	3 sec
composed-75-1-80-6_ext	0 sec	3 sec
BlackHole-4-7-h-9_ext	4 sec	32 sec
qwh-15-106-1_ext	6 sec	300 sec (5 min)
driverlogw-08c-sat_ext	47 sec	600 sec (10 min)

Όπως βλέπουμε ο αλγόριθμός Ακτινικής Αναζήτησης γίνεται όλο και πιο χρονοβόρος όσο μεγαλώνει το μέγεθος του προβλήματος και σε σύγκριση με τον αλγόριθμο Ελάχιστων Συγκρούσεων ο χρόνος εκτέλεσης του φτάνει να είναι ως και 50 φορές μεγαλύτερος.

Παρ'όλα αυτά όμως, εξετάζοντας τους πίνακες του κεφαλαίου 3 παρατηρούμε ότι ο αλγόριθμος Ακτινικής Αναζήτησης βρίσκει τις καταστάσεις με τις λιγότερες συγκρούσεις σε σύγκριση με τον αλγόριθμο Ελάχιστων Συγκρούσεων. Το μόνο πρόβλημα, στο οποίο βρίσκουν ως μερική λύση με ελάχιστες συγκρούσεις την ίδια τιμή είναι ο αλγόριθμος langford-2-4-ext, στον οποίο, όπως φαίνεται στους πίνακες του κεφαλαίου 3, έχει βρεθεί κάποιο τοπικό βέλτιστο ή στην καλύτερη περίπτωση το ολικό. Επίσης, είναι αξιοσημείωτο ότι ο αλγόριθμος της Ακτινικής Αναζήτησης παρουσιάζει μικρότερο μέσο όρο συγκρούσεων από τον αλγόριθμο Ελάχιστων Συγκρούσεων για k=2 και με μόνο 100 επαναλήψεις, ενώ ο αλγόριθμος Ελάχιστων Συγκρούσεων παρουσιάζει τον ελάχιστο μέσο όρο συγκρούσεων για τις μεγαλύτερες τιμές, restars=1000 και repeat=1000 .

Συνοψίζοντας τις παραπάνω παρατηρήσεις, καταλήγουμε στο συμπέρασμα ότι ο αλγόριθμος Ακτινικής Αναζήτησης είναι αποτελεσματικότερος από τον αλγόριθμο Ελάχιστων Συγκρούσεων. Ακόμα και αν είναι χρονοβόρος σε σύγκριση με τον αλγόριθμο Ελάχιστων Συγκρούσεων στο ίδιο επίπεδο επαναλήψεων. Παρατηρούμε ότι με μικρές τιμές k και μεσαίο αριθμό επαναλήψεων πάλι δίνει καλύτερα αποτελέσματα από αυτόν. Συγκεκριμένα , ο αλγόριθμος Ελάχιστων Συγκρούσεων χρειάζεται 10-πλάσιες επαναλήψεις, οπότε και πολύ περισσότερο χρόνο για την εκτέλεσή του. Ενδεικτικά, το πρόβλημα BlackHole-4-7-h-9_ext: στον αλγόριθμο Ελάχιστων Συγκρούσεων οι λιγότερες συγκρούσεις που επιτυγχάνονται είναι 11972 σε χρόνο εκτέλεσης του αλγορίθμου 785 δευτερόλεπτα (restars=1000,repeat=1000), ενώ ο αλγόριθμο Ακτινικής Αναζήτησης επιτυγχάνει 11379 συγκρούσεις σε 239 δευτερόλεπτα. Τα μόνο πρόβλημα στο οποίο και οι δύο είναι αποτελεσματικοί είναι το langford-2-4-ext, όπου ο αλγόριθμός Ακτινικής Αναζήτησης βρίσκει τη βέλτιστη λύση σε κάθε περίπτωση που εκτελέσαμε, σε αντίθεση με τον αλγόριθμο Ελάχιστων Συγκρούσεων.

Συμπερένουμε ότι ο αλγόριθμος Ακτινικής Αναζήτησης αποτελεί καλύτερη επιλογή για όλα τα είδη προβλημάτων. Αν και είναι πιο χρονοβόρος αλγόριθμος δίνει πολύ καλύτερες λύσεις από τον αλγόριθμο Ελάχιστων Συγκρούσεων, ακόμα και για μικρές τιμές του k και μικρό αριθμό επαναλήψεων. Οπότε, δεν αποτελεί σε καμία περίπτωση απλά μια χρονοβόρα έκδοση του αλγορίθμου Ελάχιστων Συγκρούσεων, άλλα αντιθέτως, όπως ξέρουμε και σε θεωρητικό επίπεδο, αποτελεί μια πιο εξελιγμένη μορφή, η οποία είναι πιο αποτελεσματική στην πράξη.

ΒΙΒΛΙΟΓΡΑΦΙΑ

- Stuart Russell-Peter Norvig, «Τεχνητή Νοημοσύνη, Μια Σύγχρονη Προσέγγιση»

- Βλαχάβας, Κεφαλάς, Βασιλειάδης, Κόκκορας, Σακελλαρίου , «Τεχνητή Νοημοσύνη»

- Solving Large ScaleConstraint Satisfaction and Scheduling Problems using a Heuristic Repair Method – Steven Minton,Mark D.Johnston,Andrew B. Philips,Philip Laird

ΠΑΡΑΡΤΗΜΑ

Αλγόριθμος Local-Beam Search

```
#include "stdafx.h"
#include "2-4d.h"
#include <algorithm>
#include <time.h>
#include <sstream>
using namespace std;



int main(int argc, _TCHAR* argv[])
{	time_t start,end;
	time (&start);

	string txt_dom,txt_con,txt_var,txt_result,txt_t;
char* ar_txt[4];

  //ar_txt[0]="hanoi-5_ext";
 ar_txt[0]="langford-2-4-ext";
//ar_txt[2]="geo50-20-d4-75-1_ext";
  // ar_txt[0]="qwh-15-106-1_ext";
  // ar_txt[1]="BlackHole-4-7-h-9_ext";
//ar_txt[2]="composed-75-1-80-6_ext";

//ar_txt[0]="driverlogw-08c-sat_ext";



for(int a=0;a<3;a++){

	string txt(ar_txt[a]);
	string tvar("_var.txt");
	string tdom("_dom.txt");
	string tcon("_con.txt");
	string result("_results.txt");
	string text(".txt");

for(int ep=0;ep<3;ep++){
time (&start);

	txt_var= txt+ tvar;
	txt_dom= txt+ tdom;
	txt_con= txt+ tcon;
	txt_t= txt+ text;



	Var arxeiavar(txt_var);
	arxeiavar.fill();

	Dom arxeiadom(txt_dom);
	arxeiadom.fill();

	Con arxeiacon(txt_con);
	arxeiacon.fill();
```

```
        int ****con,*max,k=50,temp,val,be,number=1000;
        int i1,i2,I1,I2,i3,i4,randomvar,**randomval,*minconflict;
        int
*randomi,*varconflicts,***conflict,*sorted,flag=0,flag1=0,flag3=0,restart=-
1,bests=1000;
        int b=0,**bestk1,***bestk,*best1,t1=-
1,**best,random,**check,minconflicts,conf=0,*values,count=0,t=0;
        double dif;
        string snum,sep;
        stringstream out,out1;

                max =(int*) malloc(arxeiavar.varn*sizeof(int));

        for (i1=0; i1<arxeiavar.varn; i1++) {
                for( int j1=0;j1<(arxeiadom.domn[arxeiavar.var[i1]]);j1++){
                        for(int
j2=0;j2<(arxeiadom.domn[arxeiavar.var[i1]]);j2++){


        if(arxeiadom.dom[arxeiavar.var[i1]][j1]>=arxeiadom.dom[arxeiavar.var[i1]
][j2])
                                {max[i1]=arxeiadom.dom[arxeiavar.var[i1]][j1];
                                }}}
        }



        con = (int ****)malloc(arxeiavar.varn*sizeof(int ***));
        for (i1=0; i1<arxeiavar.varn; i1++) {
                con[i1] = (int ***)malloc(arxeiavar.varn*sizeof(int **));
        for (i2=0; i2<arxeiavar.varn; i2++) {
                con[i1][i2]=(int**) malloc((max[i1]+1)*sizeof(int*));
        for( i3=0;i3<=max[i1];i3++){
                con[i1][i2][i3]=(int*) malloc((max[i2]+1)*sizeof(int));
        for(i4=0;i4<=max[i2];i4++) {
                con[i1][i2][i3][i4]=0;


        }}}}



        for (i1=0; i1<arxeiavar.varn ; i1++) {
        for (i2=0; i2<arxeiavar.varn; i2++) {
        for( i3=0;i3<arxeiacon.N;i3++){
                for( i4=1;i4<=arxeiacon.conn[i3];i4++){



        if((arxeiacon.con[i3][0]==i1)&&(arxeiacon.con[i3][1]==i2)){

        int flag1=0;

        for (int i=0;i<arxeiadom.domn[arxeiavar.var[i1]];i++){


        if(arxeiadom.dom[arxeiavar.var[i1]][i]==arxeiacon.con[i3][2*i4]){
                flag1=1;
        }
                }

        int flag2=0;
```

```
for (int i=0;i<arxeiadom.domn[arxeiavar.var[i2]];i++){

if(arxeiadom.dom[arxeiavar.var[i2]][i]==arxeiacon.con[i3][2*i4+1]){
	flag2=1;}
}

if (flag1==1&&flag2==1){

con[i1][i2][arxeiacon.con[i3][2*i4]][arxeiacon.con[i3][2*i4+1]]=1;}

		}



	}}}}


varconflicts=(int*) malloc(1*sizeof(int));
values=(int*) malloc(1*sizeof(int));

randomval = (int**) malloc(arxeiavar.varn*sizeof(int));
for(int i=0;i<arxeiavar.varn;i++){
	randomval[i]=(int*) malloc(k*sizeof(int));}


check = (int**) malloc(arxeiavar.varn*sizeof(int));
for(int i=0;i<arxeiavar.varn;i++){
	check[i]=(int*) malloc(k*sizeof(int));}

randomi=(int*) malloc(k*arxeiavar.varn*sizeof(int));



conflict=(int***) malloc(arxeiavar.varn*sizeof(int));
for(int i=0;i<arxeiavar.varn;i++){
conflict[i]=(int**) malloc(k*sizeof(int));
for(int j=0;j<k;j++){

	int l=arxeiadom.domn[arxeiavar.var[i]];
	conflict[i][j]=(int*) malloc(l*sizeof(int));}}

int size=0;
for(int i1=0;i1<arxeiavar.varn;i1++){
size=(arxeiavar.varn*arxeiadom.domn[arxeiavar.var[i1]])+size;}

sorted=(int*) malloc(size*sizeof(int));
int* sortedi=(int*) malloc(size*sizeof(int));
int* sortedI=(int*) malloc(size*sizeof(int));
int* sortedj=(int*) malloc(size*sizeof(int));
int* sorted1=(int*) malloc(k*k*sizeof(int));
int* sortedq=(int*) malloc(k*k*sizeof(int));
int* sortedj1=(int*) malloc(k*k*sizeof(int));

best1=(int*) malloc(k*sizeof(int));

best=(int**) malloc(k*sizeof(int));
for(int j=0;j<k;j++){
	best[j]=(int*) malloc(k*sizeof(int));}

bestk=(int***) malloc(k*sizeof(int));
for(int j=0;j<k;j++){
```

```
		bestk[j]=(int**) malloc(k*sizeof(int));
	for(int qu=0;qu<k;qu++){
		bestk[j][qu]=(int*) malloc(arxeiavar.varn*sizeof(int));}}

	bestk1=(int**) malloc(k*sizeof(int));
	for(int j=0;j<k;j++){
		bestk1[j]=(int*) malloc(arxeiavar.varn*sizeof(int));}



	srand(time(0));



	for(int i=0;i<arxeiavar.varn;i++){
			randomi[i]=0;
		for(int j=0;j<k;j++){
	randomval[i][j] = 0;
		}}

	for(int j=0;j<k;j++){

	for(int i=0;i<arxeiavar.varn;i++){

		 randomi[i]=((rand()+ep) % arxeiadom.domn[arxeiavar.var[i]]);

	}

	for(int i=0;i<arxeiavar.varn;i++){

	randomval[i][j]=arxeiadom.dom[arxeiavar.var[i]][randomi[i]];

	}}




	restart=-1;

	do{	int flag2=0;int s1=1000000;

		for(int i=0;i<arxeiavar.varn;i++){
			for(int j=0;j<k;j++){
				for(int
I=0;I<arxeiadom.domn[arxeiavar.var[i]];I++){
					check[i][j]=randomval[i][j];
						conflict[i][j][I]=0;
				}}}


		for(int j=0;j<k;j++){

	    for(int i=0;i<arxeiavar.varn;i++){
		  for(int I=0;I<arxeiadom.domn[arxeiavar.var[i]];I++){

	check[i][j]=arxeiadom.dom[arxeiavar.var[i]][I];



		for(int i1=0;i1<arxeiavar.varn;i1++){
```

```
                    for(int e=0;e<arxeiavar.varn;e++){
                                        if
(con[i1][e][check[i1][j]][check[e][j]]==0){
                                                            conflict[i][j][I]++;

                                        }
                                }}
                }
                check[i][j]=randomval[i][j];
                }}




                for(int j=0;j<k;j++){
                        int s=-1;
                        for(int i1=0;i1<arxeiavar.varn;i1++){
                                for(int
I1=0;I1<arxeiadom.domn[arxeiavar.var[i1]];I1++){
                                s++;
                                        val=conflict[i1][j][I1];
                                        sorted[s]=val; sortedi[s]=i1; sortedI[s]=I1;

                                        }}


                                for(int qu=0;qu<=s;qu++){

                                for(int qu1=0;qu1<=s;qu1++){
                                        if (sorted[qu1]>sorted[qu])
                                        {
                                                temp = sorted[qu];
                                                sorted[qu] =sorted[qu1];
                                                sorted[qu1] = temp;

                                                int temp1 = sortedi[qu];
                                                sortedi[qu] =sortedi[qu1];
                                                sortedi[qu1] = temp1;

                                                int temp2 = sortedI[qu];
                                                sortedI[qu] =sortedI[qu1];
                                                sortedI[qu1] = temp2;

                                        }}}



                                            for(int qu=0;qu<k;qu++){
                                                best[j][qu]=sorted[qu];
                                               for(int i=0;i<arxeiavar.varn;i++){

                                               if (i==sortedi[qu]){

        bestk[j][qu][i]=arxeiadom.dom[arxeiavar.var[i]][sortedI[qu]];}
                                else{bestk[j][qu][i]=randomval[i][j];}
                                        }}



                }
                 for(int qu=0;qu<k;qu++){
```

```
         for(int j=0;j<k;j++){
                     if (best[j][qu]==0){       flag1=1;b=qu;be=j;}}}




              if (flag1==0){


                     int qu6=0;
       for(int j=0;j<k;j++){
                     for(int qu=0;qu<k;qu++){

                                   val=best[j][qu];
                                   sorted1[qu6]=val; sortedq[qu6]=qu;
sortedj1[qu6]=j;
                                   qu6++;

                            }}

                            for(int qu2=0;qu2<k*k;qu2++){

                            for(int qu3=0;qu3<k*k;qu3++){

                                   if (sorted1[qu3]>sorted1[qu2])
                                   {
                                          int temp3 = sorted1[qu2];
                                          sorted1[qu2] =sorted1[qu3];
                                          sorted1[qu3] = temp3;


                                          int temp4 = sortedq[qu2];
                                          sortedq[qu2] =sortedq[qu3];
                                          sortedq[qu3] = temp4;

                                          int temp5 = sortedj1[qu2];
                                          sortedj1[qu2] =sortedj1[qu3];
                                          sortedj1[qu3] = temp5;


                                   }
                            }



                            for(int qu=0;qu<k;qu++){
                                                 val=sorted1[qu];
                                                 best1[qu]=val;
                                                 for(int
i=0;i<arxeiavar.varn;i++){


bestk1[qu][i]=bestk[sortedj1[qu]][sortedq[qu]][i];}}

                  for(int qu=0;qu<k;qu++){
                        if(best1[qu]==0){flag=1;b=qu;}
                        }
```

```
		}
		restart++;

		if(((flag==1)&&(flag1==0))||((flag==1)&&(flag1==0))){flag3=1;}

		for(int j=0;j<k;j++){
		for(int i=0;i<arxeiavar.varn;i++){
				randomval[i][j]=bestk1[j][i];
		}}
	}while((restart<number)&&(flag3==0));




	out << number;
	out1<< ep;
	snum = out.str();
	sep = out1.str();
	time (&end);
	dif = difftime (end,start);

	txt_result=txt+ snum + sep +result;
	ofstream myfile (txt_result);

	if (myfile.is_open())  {

		if(flag1==1){

				myfile<<"Brethike lusi";
				myfile<<"se "<<restart<<" restarts!kai xrono
"<<dif;
				for(int i=0;i<arxeiavar.varn;i++){


	myfile<<"bestk["<<i<<"]="<<bestk[be][b][i]<<"\n";}
			}

				else if(flag==1){
				myfile<<"Brethike lusi";
				myfile<<"se "<<restart<<" restarts!kai xrono
"<<dif;
				for(int i=0;i<arxeiavar.varn;i++){


	myfile<<"bestk1["<<i<<"]="<<bestk1[i][b]<<"\n";}

				}

				else{
				myfile<<"Exoume "<<best1[0]<< "
sigkrouseis!\n";
				myfile<<"se "<<restart<<" restarts!kai xrono
"<<dif<<"sec";
				for(int i=0;i<arxeiavar.varn;i++){

	myfile<<"bestk1["<<i<<"]="<<bestk1[0][i]<<"\n";
					}
```

```
                                        }
        }

        myfile.close();}}
        system("pause");
        return 0;

}
```

Αλγόριθμος Min-conflicts

```
#include "stdafx.h"
#include "2-4d.h"
#include <sstream>
using namespace std;


int main(int argc, _TCHAR* argv[])
{       time_t start,end;



        string txt_dom,txt_con,txt_var,txt_result,txt_t;

        char* ar_txt[4];

  //ar_txt[2]="hanoi-5_ext";
//ar_txt[1]="langford-2-4-ext";
 //ar_txt[0]="geo50-20-d4-75-1_ext";
  //ar_txt[0]="qwh-15-106-1_ext";
   ar_txt[0]="BlackHole-4-7-h-9_ext";
   //ar_txt[1]="composed-75-1-80-6_ext";

//ar_txt[0]="driverlogw-08c-sat_ext";



for(int a=0;a<3;a++){

        string txt(ar_txt[a]);
        string tvar("_var.txt");
        string tdom("_dom.txt");
        string tcon("_con.txt");
        string result("_results.txt");
        string text(".txt");

for(int ep=0;ep<3;ep++){
time (&start);

        txt_var= txt+ tvar;
        txt_dom= txt+ tdom;
        txt_con= txt+ tcon;
        txt_t= txt+ text;



        Var arxeiavar(txt_var);
        arxeiavar.fill();

        Dom arxeiadom(txt_dom);
```

```
        arxeiadom.fill();

        Con arxeiacon(txt_con);
        arxeiacon.fill();

        double dif;
        int ****con,*max;
        int i1,i2,i3,i4,randomvar,*randomval,number=1000;
        int *randomi,*varconflicts,*conflict,flag=0,flag1=0,restart=-1;
        int b=0,best=0,t1=-
1,*bests,random,*checked,minconflicts,conf=0,*values,count=0;
        string snum,sep;
        stringstream out,out1;

        free(arxeiacon.x);

               max =(int*) malloc(arxeiavar.varn*sizeof(int));

        for (i1=0; i1<arxeiavar.varn; i1++) {
               for( int j1=0;j1<(arxeiadom.domn[arxeiavar.var[i1]]);j1++){
                      for(int
j2=0;j2<(arxeiadom.domn[arxeiavar.var[i1]]);j2++){


        if(arxeiadom.dom[arxeiavar.var[i1]][j1]>=arxeiadom.dom[arxeiavar.var[i1]
][j2])
                              {max[i1]=arxeiadom.dom[arxeiavar.var[i1]][j1];
                              }}}
        }



        con = (int ****)malloc(arxeiavar.varn*sizeof(int ***));
        for (i1=0; i1<arxeiavar.varn; i1++) {
               con[i1] = (int ***)malloc(arxeiavar.varn*sizeof(int **));
        for (i2=0; i2<arxeiavar.varn; i2++) {
               con[i1][i2]=(int**) malloc((max[i1]+1)*sizeof(int*));
        for( i3=0;i3<=max[i1];i3++){
               con[i1][i2][i3]=(int*) malloc((max[i2]+1)*sizeof(int));
        for(i4=0;i4<=max[i2];i4++) {
               con[i1][i2][i3][i4]=0;


        }}}}



for (i1=0; i1<arxeiavar.varn ; i1++) {
        for (i2=0; i2<arxeiavar.varn; i2++) {
        for( i3=0;i3<arxeiacon.N;i3++){
               for( i4=1;i4<=arxeiacon.conn[i3];i4++){



        if((arxeiacon.con[i3][0]==i1)&&(arxeiacon.con[i3][1]==i2)){

        int flag1=0;

        for (int i=0;i<arxeiadom.domn[arxeiavar.var[i1]];i++){


        if(arxeiadom.dom[arxeiavar.var[i1]][i]==arxeiacon.con[i3][2*i4]){
```

```
            flag1=1;
      }
            }

      int flag2=0;

      for (int i=0;i<arxeiadom.domn[arxeiavar.var[i2]];i++){

      if(arxeiadom.dom[arxeiavar.var[i2]][i]==arxeiacon.con[i3][2*i4+1]){
            flag2=1;}
      }

      if (flag1==1&&flag2==1){

      con[i1][i2][arxeiacon.con[i3][2*i4]][arxeiacon.con[i3][2*i4+1]]=1;}

                  }



            }}}}

      varconflicts=(int*) malloc(1*sizeof(int));
      values=(int*) malloc(1*sizeof(int));
      randomval = (int*) malloc(arxeiavar.varn*sizeof(int));
      randomi=(int*) malloc(arxeiavar.varn*sizeof(int));
      checked=(int*) malloc(arxeiavar.varn*sizeof(int));
      conflict=(int*) malloc(arxeiavar.varn*sizeof(int));
      bests=(int*) malloc(arxeiavar.varn*sizeof(int));
       srand(time(0));
do{


      b=0;



      for(int i=0;i<arxeiavar.varn;i++){
      randomval[i] = 0;
      randomi[i]=0;
      }


      for(int i=0;i<arxeiavar.varn;i++){

              randomi[i]=((rand()+ep)% arxeiadom.domn[arxeiavar.var[i]]);

      }

      for(int i=0;i<arxeiavar.varn;i++){

      randomval[i]=arxeiadom.dom[arxeiavar.var[i]][randomi[i]];

      }



      do{


            for(int i=0;i<arxeiavar.varn;i++){
                  conflict[i]=0;
```

```
		}


		int t=0;
		for(int i=0;i<arxeiavar.varn;i++){
			for(int j=0;j<arxeiavar.varn;j++){
				if (con[i][j][randomval[i]][randomval[j]]==0){
				conflict[i]++;}
				else
				{ t++;}
		}}



		if (t == power(arxeiavar.varn,2)){
			cout<<"To provlima luthike!!!!" ;
			flag=1;}

		else{

			do{
				random=rand()%(arxeiavar.varn);


			}while ((conflict[random]==0 ));


			if (t>best){ for(int
i=0;i<arxeiavar.varn;i++){bests[i]=randomval[i];

			 }
			best=t;
			}



			varconflicts=(int*)
realloc(varconflicts,arxeiadom.domn[arxeiavar.var[random]]*sizeof(int));
			values=(int*)
realloc(values,arxeiadom.domn[arxeiavar.var[random]]*sizeof(int));

			for(int z=0;z<arxeiadom.domn[arxeiavar.var[random]];z++){
				varconflicts[z]=0;}

			for(int z=0;z<arxeiadom.domn[arxeiavar.var[random]];z++){
				for(int j=0;j<arxeiavar.varn;j++){
					if
(con[random][j][arxeiadom.dom[arxeiavar.var[random]][z]][randomval[j]]==0){
					varconflicts[z]++;
					}
				}}



			minconflicts=varconflicts[-1]=10000;

			for(int i=0;i<arxeiadom.domn[arxeiavar.var[random]];i++){
				if(varconflicts[i]<varconflicts[i-
1]){minconflicts=varconflicts[i];  }
				}
```

```
                        count=0;
                        for(int i=0;i<arxeiadom.domn[arxeiavar.var[random]];i++){
                               if(varconflicts[i]==minconflicts){
                                      values[count]=i;
                                      count++;
                               }
                        }

                        int r=(rand()+ep)%count;


                        randomval[random]=values[r];



                        b++;

                        }
                        }while((flag==0)&&(b<=(1000)));



                restart++;


                }while((restart<number)&&flag==0);

                out << number;
                out1<< ep;
                snum = out.str();
                sep = out1.str();
                time (&end);
                dif = difftime (end,start);

        txt_result=txt +snum + sep + result;
        ofstream file (txt_result);

        if (file.is_open())  {

                file<<"\n Se "<<b<<" epanalipseis!!!!!! kai "<<restart<<"restarts
\n";
                file<<" Se xrono "<<dif<<" sec      ";
                if (flag==1){
                for(int i=0;i<arxeiavar.varn;i++){
                        file<<"H timi tou "<<i<<" einai "<<randomval[i];
                        file<<" kai exei conflicts "<<conflict[i]<<"\n";
                }}
                else{
                        file<<"Me  "<<power(arxeiavar.varn,2)-best<<"  sigkrouseis
\n";
                        file<<"H veltisti lusi einai \n";

                        for(int i=0;i<arxeiavar.varn;i++){

                        file<<"bests["<<i<<"]="<<bests[i]<<"\n";
                                }}

                file.close();}


}}
```

```
system("pause");
return 0;}
```