ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
& ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

# ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

---

# ΕΠΙΛΥΣΗ ΠΡΟΒΛΗΜΑΤΩΝ ΙΚΑΝΟΠΟΙΗΣΗΣ ΠΕΡΙΟΡΙΣΜΩΝ ΣΥΝΔΥΑΖΟΝΤΑΣ SIMULATED ANNEALING ΚΑΙ ΔΙΑΔΟΣΗ ΠΕΡΙΟΡΙΣΜΩΝ

---

Γουλιός Σωκράτης-Παναγιώτης

Υπό την επίβλεψη και καθοδήγηση του καθηγητή

Στεργίου Κωνσταντίνου

# Περίληψη

Ένας σημαντικός κλάδος, στον οποίον βρίσκει εφαρμογή η Τεχνητή Νοημοσύνη, είναι η επίλυση προβλημάτων αναζήτησης. Στην κατηγορία αυτή υπάγονται τα Προβλήματα Ικανοποίησης Περιορισμών (Constraint Satisfaction Problems -CSPs). Ένα πρόβλημα ικανοποίησης περιορισμών απαιτεί μια τιμή, επιλεγμένη από ένα δεδομένο πεπερασμένο πεδίο, να εκχωρηθεί σε κάθε μεταβλητή του προβλήματος, έτσι ώστε να ικανοποιούνται όλοι οι περιορισμοί που σχετίζονται με τις μεταβλητές. Η επίλυση των προβλημάτων αυτών επιτυγχάνεται με χρήση της αναζήτησης και της διάδοσης περιορισμών (Constraint Propagation). Οι αλγόριθμοι που εφαρμόζουν την συνέπεια τόξου (Arc Consistency) και ο αλγόριθμος της προσομοιωμένης ανόπτησης (Simulated Annealing) χρησιμοποιούνται για την επίλυση τέτοιου είδους προβλημάτων.

Σκοπός της συγκεκριμένης διπλωματικής εργασίας είναι η μελέτη και η υλοποίηση των μεθόδων της συνέπειας τόξου και της προσομοιωμένης ανόπτησης. Πιο συγκεκριμένα για την συνέπεια, μελετήθηκε ο αλγόριθμος AC-3 (Arc Consistency Algorithm #3). Υλοποιήθηκαν και εξετάστηκαν επίσης πιθανοί συνδυασμοί τους με σκοπό τον σχεδιασμό και την υλοποίηση υβριδικών αλγορίθμων για την επίλυση προβλημάτων ικανοποίησης περιορισμών, έχοντας ως κριτήριο την άμεση ή έμμεση επιρροή της συνέπειας τόξου στην ανάθεση τιμών στις μεταβλητές από τον αλγόριθμο της προσομοιωμένης ανόπτησης.

Λέξεις Κλειδιά: Προβλήματα Ικανοποίησης Περιορισμών, Διάδοση Περιορισμών, Συνέπεια Τόξου, Προσομοιωμένη Ανόπτηση

# Abstract

An important field in which Artificial Intelligence is applied is solving search problems. This area of AI includes Constraint Satisfaction Problems (CSPs). A constraint satisfaction problem requires a value, selected from a given finite domain, to be assigned to each variable in the problem, so that all constraints relating the variables are satisfied . Solving these problems is achieved through search and Constraint Propagation. Arc Consistency and Simulated Annealing algorithms are used to solve such problems.

The purpose of this paper is the study and implementation of Arc Consistency and Simulated Annealing methods. More specifically for Arc Consistency, the AC-3 algorithm (Arc Consistency Algorithm #3) was studied. Their possible combinations were also implemented and examined in order to design and implement hybrid algorithms for solving constraint satisfaction problems, having as a criterion the direct or indirect influence of Arc Consistency on the assignment of values to the variables by the Simulated Annealing algorithm.

Keywords: Constraint Satisfaction Problems, Constraint Propagation, Arc Consistency, Simulated Annealing

## Δήλωση Πνευματικών Δικαιωμάτων

**Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν. 1599/1986 και τα άρθρα 2,4,6 παρ. 3 του Ν. 1256/1982, η παρούσα Διπλωματική Εργασία με τίτλο**

**"ΕΠΙΛΥΣΗ ΠΡΟΒΛΗΜΑΤΩΝ ΙΚΑΝΟΠΟΙΗΣΗΣ ΠΕΡΙΟΡΙΣΜΩΝ ΣΥΝΔΥΑΖΟΝΤΑΣ SIMULATED ANNEALING ΚΑΙ ΔΙΑΔΟΣΗ ΠΕΡΙΟΡΙΣΜΩΝ "**

**καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας και αναφέρονται ρητώς μέσα στο κείμενο που συνοδεύουν, και η οποία έχει εκπονηθεί στο Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Πανεπιστημίου Δυτικής Μακεδονίας, υπό την επίβλεψη του μέλους του Τμήματος κ. Κωνσταντίνου Στεργίου αποτελεί αποκλειστικά προϊόν προσωπικής εργασίας και δεν προσβάλλει κάθε μορφής πνευματικά δικαιώματα τρίτων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή / και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή.**



**Υπογραφή Φοιτητή:**

# Περιεχόμενα

# Κεφάλαιο 1

# Εισαγωγή

## 1.1 Κίνητρο

Κίνητρο για την υλοποίηση της εργασίας είναι η μελέτη μεθόδων επίλυσης προβλημάτων ικανοποίησης περιορισμών. Πιο συγκεκριμένα, μελετώνται η συνέπεια τόξου και ο Simulated Annealing, καθώς και η ανάπτυξη υβριδικών αλγορίθμων για την επίλυση τους.

## 1.2 Περιγραφή Προβλημάτων Ικανοποίησης Περιορισμών

Τα προβλήματα ικανοποίησης περιορισμών εφαρμόζονται σε πολλά παραδείγματα στην καθημερινότητα. Η αποτελεσματική κατανομή πόρων, η δρομολόγηση οχημάτων και ο σχεδιασμός ενεργειών είναι μερικά παραδείγματα. Αποτελούνται από ένα σύνολο από μεταβλητές που πρέπει να πάρουν τιμές υπό την επιβολή ενός συνόλου περιορισμών. Υπάρχουν αρκετές τεχνικές επίλυσης προβλημάτων ικανοποίησης περιορισμών, όμως στην παρούσα εργασία επικεντρωνόμαστε σε αλγορίθμους που εφαρμόζουν τη συνέπεια τόξου και στον Simulated Annealing ως μεθόδους επίλυσης.

## 1.3 Οργάνωση Διπλωματικής

Η διπλωματική εργασία έχει οργανωθεί ως εξής. Στο Κεφάλαιο 2 αναλύονται τα προβλήματα ικανοποίησης περιορισμών και αναφέρονται τεχνικές επίλυσης τους, οι αλγόριθμοι επιδιόρθωσης και οι αλγόριθμοι υπαναχώρησης. Στο κεφάλαιο 3 εξηγείται η συνέπεια τόξου, ο ψευδοκώδικας του αλγορίθμου AC-3 (Arc Consistency Algorithm #3) και γίνονται αναφορές στους υπόλοιπους αλγορίθμους. Στο Κεφάλαιο 4 γίνεται η εισαγωγή στον

Simulated Annealing και εξηγείται ο ψευδοκώδικας του. Στο Κεφάλαιο 5 αναλύονται οι υβριδικοί αλγόριθμοι των Arc Consistency και Simulated Annealing και στο Κεφάλαιο 6 παρουσιάζονται τα πειραματικά αποτελέσματα.

# Κεφάλαιο 2

# Προβλήματα Ικανοποίησης Περιορισμών

## 2.1 Εισαγωγή στα Προβλήματα Ικανοποίησης Περιορισμών

Τα Προβλήματα Ικανοποίησης Περιορισμών (Constraint Satisfaction Problems, CSPs) περιέχουν ένα σύνολο από μεταβλητές που πρέπει να πάρουν τιμές υπό την επιβολή ενός συνόλου περιορισμών **[1]**. Μπορούν να οριστούν αλγόριθμοι αναζήτησης που εκμεταλλεύονται την δομή των καταστάσεων και χρησιμοποιούν ευρετικούς μηχανισμούς γενικής χρήσης ώστε να είναι εφικτή η επίλυση μεγάλων προβλημάτων. Τα προβλήματα ικανοποίησης περιορισμών αποτελούν αντικείμενο μελέτης της τεχνητής νοημοσύνης.

Γενικότερα, κάθε πρόβλημα ικανοποίησης περιορισμών περιέχει:

- Ένα σύνολο n μεταβλητών $V_1,V_2,\ldots,V_n$
- Ένα σύνολο n πεδίων τιμών $D_1,D_2,\ldots,D_n$ που αντιστοιχούν σε κάθε μεταβλητή
- Ένα σύνολο m περιορισμών $C_1,C_2,\ldots,C_m$ , όπου $C_i$ αναφέρεται σε κάποιο υποσύνολο των μεταβλητών και καθορίζει το σύνολο των επιτρεπτών συνδυασμών τιμών

Αντίστοιχα, οι περιορισμοί διακρίνονται σε:

- Μοναδιαίοι(Unary), περιορισμοί που επιβάλλονται σε μια μεταβλητή (για παράδειγμα, X>3)

- Δυαδικοί(Binary), περιορισμοί που επιβάλλονται σε δύο μεταβλητές (για παράδειγμα, X<Y)
- Ανώτερης τάξης(higher order) ή n-αδικοί, περιορισμοί που επιβάλλονται σε τρεις ή περισσότερες μεταβλητές(για παράδειγμα, X+Y > Z)

Μια πλήρης ανάθεση τιμών στις μεταβλητές που είναι συνεπής (κανένας περιορισμός δεν παραβιάζεται) και ολοκληρωμένη (έχουν ανατεθεί σε όλες τις μεταβλητές τιμές) αποτελεί λύση του προβλήματος **[2]**. Τα CSPs ανήκουν στην κατηγορία NP-Complete, προβλήματα που στην χειρότερη περίπτωση επιλύονται σε εκθετικό χρόνο και η λύση τους επαληθεύεται πολυωνυμικά.

Απλά παραδείγματα αποτελούν το πρόβλημα με τις 8 βασίλισσες στην σκακιέρα (λύση αποτελεί η τοποθέτηση των βασιλισσών με σκοπό να μην απειλεί η μία την άλλη), ο "χρωματισμός" γραφημάτων (λύση αποτελεί η ανάθεση χρωμάτων στις κορυφές ώστε να μην υπάρχουν γειτονικές κορυφές με το ίδιο χρώμα) και το Sudoku (σωστή τοποθέτηση αριθμών έτσι ώστε ένας αριθμός να εμφανίζεται μια φορά στην γραμμή, στο block και στην στήλη που ανήκει). Στην καθημερινότητα τα CSPs εφαρμόζονται στην αποτελεσματική κατανομή πόρων, στην αυτοματοποίηση διαδικασιών, στην χωρική και χρονική συλλογιστική, στην κατανομή συχνοτήτων και σε πολλά άλλα συνδυαστικά προβλήματα **[3]**.

## 2.2 Αλγόριθμοι Επιδιόρθωσης

Οι αλγόριθμοι επιδιόρθωσης (Repair Algorithms) ή μέθοδοι τοπικής αναζήτησης (Local Search Methods) δημιουργούν μια ανάθεση τιμών στις μεταβλητές και αλλάζουν επαναληπτικά τις τιμές τους με στόχο να αυξήσουν τον αριθμό των περιορισμών που ικανοποιούνται. Μπορούν να βρουν λύσεις σε προβλήματα, αλλά η πιθανότητα να "κολλήσουν" σε τοπικά βέλτιστα τους καθιστά ατελείς.

### 2.2.1 Αλγόριθμος Αναρρίχησης Λόφου

Ο αλγόριθμος Αναρρίχησης Λόφου (Hill Climbing - HC) είναι ένας αλγόριθμος τοπικής αναζήτησης. Είναι ένας βρόγχος που μετακινείται συνεχώς στην κατεύθυνση της αυξανόμενης τιμής μιας συνάρτησης βελτιστοποίησης και τερματίζει όταν φτάσει σε μια "κορυφή" όπου καμία γειτονική κατάσταση δεν έχει καλύτερη τιμή από την τρέχουσα κατάσταση. Δεν συντηρεί δένδρο αναζήτησης, γι'αυτό στην δομή δεδομένων του τρέχοντος κόμβου χρειάζεται να καταγράφεται μόνο η κατάσταση και η τιμή της αντικειμενικής συνάρτησης. Η αναρρίχηση λόφων δεν κοιτάζει πιο πέρα από τους άμεσους γείτονες της τρέχουσας κατάστασης **[2]**.

Πλεονέκτημα του Hill Climbing είναι ότι εξετάζει ένα μεγάλο πλήθος γειτονικών καταστάσεων, βασικό του μειονέκτημα όμως αποτελεί η πιθανότητα να "κολλήσει" σε τοπικό μέγιστο ή ελάχιστο, ανάλογα το πρόβλημα που εξετάζει **[4]**.

Παρακάτω παρουσιάζεται ο ψευδοκώδικας του αλγορίθμου Hill Climbing για CSPs

Hill Climbing Algorithm

Είσοδος

CSP, ΑρχικήΑνάθεση

Έξοδος

ΤελικήΑνάθεση

1. ΤωρινήΑνάθεση = ΑρχικήΑνάθεση
2. Στόχος = False
3. **WHILE** Στόχος = **false**

1. ΓειτονικήΑνάθεση = Η καλύτερη επιλογή από τις γειτονικές καταστάσεις
2. **IF** η ΓειτονικήΑνάθεση δεν είναι καλύτερη από την Τωρινή
    i. Στόχος = True
3. **ELSE**
    i. ΤωρινήΑνάθεση = ΓειτονικήΑνάθεση
4. **ΤΕΛΟΣ_IF**

4. **ΤΕΛΟΣ_WHILE**
5. ΤελικήΑνάθεση = ΤωρινήΑνάθεση
6. **ΕΠΕΣΤΡΕΨΕ** ΤελικήΑνάθεση

Στην αρχή του αλγορίθμου αναθέτονται τυχαίες τιμές σε όλες τις μεταβλητές από τα αντίστοιχα πεδία ορισμού τους. Οι γειτονικές καταστάσεις διαφέρουν από την τρέχουσα κατά μία ανάθεση τιμής σε μεταβλητή. Η καλύτερη γειτονική κατάσταση, η οποία επιλέγεται με κριτήριο το πλήθος των περιορισμών που ικανοποιούνται, επιλέγεται ως γειτονική ανάθεση και συγκρίνεται με την τρέχουσα. Η τωρινή ανάθεση αποθηκεύει αρχικά την αρχική ανάθεση και μετέπειτα τις πιθανές βελτιωμένες γειτονικές αναθέσεις.

Ο συγκεκριμένος αλγόριθμος αναρρίχησης ονομάζεται Steepest Ascent Hill Climbing (SAHC). Μια διαφορετική παραλλαγή του αλγορίθμου αναρρίχησης αποτελεί ο Στοχαστικός Αλγόριθμος Αναρρίχησης (Stochastic Hill Climbing), ο οποίος επιλέγει αν θα μετακινηθεί ο αλγόριθμος από την ΤωρινήΑνάθεση στην ΓειτονικήΑνάθεση με κριτήριο την βελτίωση που παρουσιάζει.

### 2.2.2 Ευρετικός Αλγόριθμος Ελαχίστων Συγκρούσεων

Ο Ευρετικός Αλγόριθμος Ελαχίστων Συγκρούσεων (Minimum Conflicts Heuristic) αρχικά αναθέτει τυχαίες τιμές στις μεταβλητές από τα πεδία ορισμών τους, έπειτα ελέγχεται αν αυτή η ανάθεση αποτελεί λύση, δηλαδή να μην παραβιάζεται κανένας περιορισμός. Αν δεν αποτελεί λύση, επιλέγεται τυχαία μια μεταβλητή που παραβιάζει κάποιον περιορισμό και εξετάζει όλες τις πιθανές τιμές της. Επιλέγεται ως νέα τιμή της μεταβλητής η τιμή, η οποία με την εφαρμογή της επιτυγχάνει την μέγιστη μείωση των περιορισμών που

παραβιάζονται. Έπειτα, ξαναεξετάζεται αν η νέα ανάθεση αποτελεί λύση του προβλήματος **[5]**.

Σε αντίθεση με τον αλγόριθμο Hill Climbing, ο Ευρετικός Αλγόριθμος Ελαχίστων Συγκρούσεων δεν εξετάζει μεγάλο πλήθος γειτονικών καταστάσεων, ενώ είναι και αυτός επιρρεπής να "κολλήσει" σε τοπικό ελάχιστο.

Παρακάτω παρουσιάζεται ο ψευδοκώδικας του αλγορίθμου Minimum Conflicts Algorithm

## Minimum Conflicts Algorithm

## Είσοδος

CSP, ΑρχικήΑνάθεση, Max_Steps

## Έξοδος

Μια λύση για το CSP ή αποτυχία

1. ΤρέχουσαΑνάθεση = Αρχική πλήρης ανάθεση τιμών
2. **FOR** i από 1 μέχρι Max_Steps
    1. **IF** η ΤρέχουσαΑνάθεση αποτελεί λύση του CSP
        i. **ΕΠΕΣΤΡΕΨΕ** ΤρέχουσαΑνάθεση
    2. **ELSE**
        i. Μεταβλητή = Τυχαία Μεταβλητή που επιλέγεται από το σύνολο μεταβλητών που παραβιάζουν περιορισμούς
        ii. Τιμή = η τιμή v της μεταβλητής που ελαχιστοποιεί την CONFLICTS(Μεταβλητή.v, ΤρέχουσαΑνάθεση,CSP)
        iii. Μεταβλητή = Τιμή
    3. **ΤΕΛΟΣ_IF**
3. **ΤΕΛΟΣ_FOR**
4. **Τερματισμός Αλγορίθμου(return failure)**

Η αρχική ανάθεση μπορεί να επιλέγεται τυχαία ή με μια διαδικασία άπληστης ανάθεσης τιμών που επιλέγει μια τιμή ελάχιστων συγκρούσεων για

την κάθε μεταβλητή. Τα Max_Steps είναι ένας προεπιλεγμένος αριθμός επιτρεπόμενων επαναλήψεων πριν εγκαταλείψει ο αλγόριθμος την προσπάθεια πλήρους ανάθεσης τιμών. Η βοηθητική συνάρτηση CONFLICTS υπολογίζει τον αριθμό των περιορισμών που παραβιάζονται από μια συγκεκριμένη τιμή, με δεδομένη την υπόλοιπη τρέχουσα ανάθεση τιμών.

## 2.3 Αλγόριθμοι Υπαναχώρησης

### 2.3.1 Απλή ή Χρονολογική Υπαναχώρηση

Η Απλή ή Χρονολογική Υπαναχώρηση (Simple or Chronological Backtracking – BT) επιλέγει τιμές για μία μόνο μεταβλητή τη φορά και υπαναχωρεί όταν μια μεταβλητή δεν έχει άλλες νόμιμες τιμές που μπορούν να της ανατεθούν **[2]**. Είναι πλήρης με πολυπλοκότητα χρόνου $O(d^n e)$ και χώρου O(nd), όπου d το μέγεθος του μεγαλύτερου πεδίου ορισμού μιας μεταβλητής, n ο αριθμός των μεταβλητών και e ο αριθμός των περιορισμών.

Ο ευρετικός μηχανισμός των ελαχίστων απομενουσών τιμών (minimum remaining values – MRV) επιλέγει την μεταβλητή με τις λιγότερες πιθανές αποδεκτές τιμές, ελαχιστοποιεί τον παράγοντα διακλάδωσης και παρέχει δυναμική επιλογή μεταβλητών στον αλγόριθμο **[6]**.

Ο ευρετικός μηχανισμός βαθμού (degree heuristic) επιλέγει την μεταβλητή που εμπλέκεται με τον μεγαλύτερο αριθμό περιορισμών με μεταβλητές που δεν τις έχουν ανατεθεί τιμές, επιλέγεται δηλαδή ο συνδυασμός μεταβλητής-κόμβου με το μεγαλύτερο βαθμό στο γράφημα των περιορισμών (η επιλογή είναι στατική). Στο γράφημα περιορισμών, οι κόμβοι αντιστοιχούν σε μεταβλητές του προβλήματος και οι ακμές (τόξα) αντιστοιχούν σε περιορισμούς. Βαθμός κόμβου σε ένα γράφημα ονομάζεται το πλήθος των ακμών που πρόσκεινται σε έναν κόμβο. Γενικότερα ο MRV είναι ισχυρότερος μηχανισμός από τον degree heuristic αλλά μπορούν να χρησιμοποιηθούν συνδυαστικά, για παράδειγμα όταν ο MRV δεν μπορεί να ξεχωρίσει δύο μεταβλητές παρεμβαίνει ο degree heuristic **[6]**.

Σε αντίθεση με τις δύο προηγούμενες μεθόδους που αναφέρθηκαν, ο ευρετικός μηχανισμός της λιγότερο δεσμεύουσας τιμής ( Least-Constraining

Value – LCV) δεν επιλέγει μεταβλητές, αλλά αποφασίζει με ποια σειρά θα εξεταστούν οι τιμές **[6]**. Για κάθε μεταβλητή επιλέγεται η τιμή που αποκλείει τις λιγότερες επιλογές τιμών από τις γειτονικές μεταβλητές του γραφήματος περιορισμών. Ο LCV παρέχει ευελιξία για τις επόμενες αναθέσεις μεταβλητών αλλά δεν είναι χρήσιμος αν το πρόβλημα δεν έχει λύση ή αν πρέπει να βρεθούν όλες οι λύσεις του.

### 2.3.2 Υπαναχώρηση με άλμα

Η υπαναχώρηση με άλμα (Backjumping – BJ) ανήκει στους αλγόριθμους έξυπνης υπαναχώρησης, αλγόριθμοι που βελτιώνουν την απλή Υπαναχώρηση. Η απλή υπαναχώρηση υποφέρει από το φαινόμενο του “trashing”, μιας κατάστασης στην οποία ο αλγόριθμος επισκέπτεται ξανά περιοχές αναζήτησης που έχει ήδη επισκεφθεί **[7]**. Η υπαναχώρηση με άλμα βελτιώνει την απλή υπαναχώρηση με την διαφορετική οπισθοδρόμηση του. Όταν φτάσει σε αδιέξοδο, ο αλγόριθμος οπισθοδρομεί στην πιο βαθιά μεταβλητή στο δέντρο αναζήτησης που συγκρούεται με την τωρινή μεταβλητή.

### 2.3.3 Πρώιμος Έλεγχος

Ο Πρώιμος Έλεγχος (Forward Checking – FC) ανήκει στην κατηγορία των look-ahead αλγορίθμων. Στοχεύει στην ικανοποίηση της εξής συνθήκης: όταν ανατίθεται τιμή σε μια μεταβλητή Χ, εξετάζεται κάθε μεταβλητή Υ που δεν της έχει ανατεθεί τιμή και συνδέεται με την Χ με έναν περιορισμό, και διαγράφεται από το πεδίο της Υ οποιαδήποτε τιμή που είναι ασυνεπής με την τιμή που ανατέθηκε στην Χ **[8]**. Ο αλγόριθμος χρησιμοποιείται συνήθως με τον Ευρετικό Μηχανισμό των ελαχίστων απομενουσών τιμών (MRV).

## 2.4 Διάδοση Περιορισμών

Μέθοδοι Διάδοσης Περιορισμών (Constraint Propagation) ονομάζονται οι μέθοδοι τροποποίησης προβλημάτων ικανοποίησης περιορισμών. Συντελούν στην διάσπαση προβλημάτων σε μικρότερα υποπροβλήματα που είναι ευκολότερα στην επίλυση τους. Στόχος είναι να λαμβάνονται υπόψιν οι

περιορισμοί νωρίς στην αναζήτηση ή ακόμα και πριν αρχίσει με την εφαρμογή τοπικής συνέπειας. Ένας τρόπος για να επιτευχθεί αυτό είναι σε κάθε βήμα της αναζήτησης να “κλαδεύονται” τμήματα του χώρου αναζήτησης και να ελέγχονται συνέπειες μερικών αναθέσεων. Οι πιο συνηθισμένες μέθοδοι είναι η συνέπεια τόξου (Arc Consistency) και η συνέπεια μονοπατιού (Path Consistency) **[10]**.

# Κεφάλαιο 3

# Συνέπεια Τόξου

## 3.1 Εισαγωγή στην Συνέπεια Τόξου

Η συνέπεια τόξου είναι μια ιδιότητα που όρισε ο Alan Mackworth το 1977 με σκοπό την απλούστευση, και σε ορισμένες περιπτώσεις την επίλυση, προβλημάτων ικανοποίησης περιορισμών. Στην γενική της έννοια, ορίζει ότι μια μεταβλητή x είναι arc consistent με κάθε άλλη μεταβλητή y, όταν για κάθε τιμή a του πεδίου ορισμού της μεταβλητής x υπάρχει τουλάχιστον μια τιμή b στο πεδίο ορισμού της μεταβλητής y, η οποία επαληθεύει τον περιορισμό μεταξύ των x και y.

Οι αλγόριθμοι που εφαρμόζουν την συνέπεια τόξου διαχειρίζονται τις μεταβλητές του προβλήματος (variables), τα πεδία ορισμών τους (domains) και τους περιορισμούς μεταξύ τους (constraints) **[11]**. Οι περιορισμοί αποθηκεύονται σε μια ουρά ή λίστα. Όταν οι περιορισμοί προστίθενται στην ουρά για να τους διαχειριστούμε, πρέπει να ληφθούν υπόψιν και οι αντίστροφοι τους. Για παράδειγμα, αν για τις μεταβλητές X και Y ισχύει ο περιορισμός X>Y ή X=Y πρέπει να προστεθεί και ο περιορισμός Y<X ή Y=X αντίστοιχα. Οι περιορισμοί αυτοί αναφέρονται συχνά και ως τόξα (Arcs). Η εφαρμογή της συνέπειας τόξου στην Υπαναχώρηση (Backtracking) πριν αρχίσει η αναζήτηση και μετά από κάθε ανάθεση τιμής επαναληπτικά συντελεί στην δημιουργία του υβριδικού αλγορίθμου Maintaining Arc Consistency(MAC) **[9]**.

## 3.2 Αλγόριθμος AC-3

Ο αλγόριθμος AC-3 (Arc Consistency Algorithm #3) είναι ο 3ος αλγόριθμος που αναπτύχθηκε από τον Alan Mackworth. Εξετάζεται κάθε φορά ένας περιορισμός. Ελέγχεται για κάθε τιμή του πεδίου ορισμού μιας μεταβλητής Α αν ικανοποιεί έναν μοναδιαίο περιορισμό (για παράδειγμα, οι τιμές του πεδίου ορισμού της να είναι θετικοί ή αρνητικοί αριθμοί) ή αν υπάρχει τιμή στο πεδίο ορισμού μιας άλλης μεταβλητής Β που γίνεται η σύγκριση. Αν δεν υπάρχει τιμή που επαληθεύει τον περιορισμό τότε διαγράφεται η τιμή από το πεδίο ορισμού της Α.

Παρακάτω παρουσιάζεται ο ψευδοκώδικας του αλγορίθμου AC-3. Η πολυπλοκότητα χρόνου του είναι $O(ad^3)$ και η πολυπλοκότητα χώρου είναι O(a), όπου το a είναι ο αριθμός των τόξων και d είναι το μέγεθος του μεγαλύτερου πεδίου ορισμού **[2][11]**.

## AC-3

## Είσοδος

Μεταβλητές Μ, Πεδίο Ορισμού ΠΟ(x) για κάθε μεταβλητή x

Μοναδιαίοι Περιορισμοί ΜΠ(x) για την μεταβλητή x που πρέπει να ικανοποιούνται

Δυαδικοί Περιορισμοί ΔΠ(x,y) για τις μεταβλητές x και y που πρέπει να ικανοποιούνται

## Έξοδος

Arc Consistent πεδία ορισμού για κάθε μεταβλητή ή πρόωρος τερματισμός του αλγορίθμου αν το πεδίο ορισμού κάποιας μεταβλητής είναι άδειο

1. **FOR** κάθε μεταβλητή Μ που έχει Μοναδιαίο Περιορισμό
    1. **FOR** κάθε τιμή x
        i. **IF** δεν ικανοποιείται ο Μοναδιαίος Περιορισμός
            1. ΠΟ(Μ) = ΠΟ(Μ) - x
        ii. **ΤΕΛΟΣ_IF**
    2. **ΤΕΛΟΣ_FOR**
2. **ΤΕΛΟΣ_FOR**
3. ΛΠ (Λίστα Περιορισμών) = Άδεια
4. **FOR** κάθε Δυαδικό Περιορισμό (x,y)
    1. ΛΠ = ΛΠ + (x,y) + (y,x)
5. **ΤΕΛΟΣ_FOR**
6. **DO**
    1. Επέλεξε ένα τόξο (x,y) από ΛΠ
    2. ΛΠ = ΛΠ - (x,y)
    3. **IF arc-reduce(x,y) = true**
        i. **IF** ΠΟ(x) είναι άδειο
            1. **Τερματισμός Αλγορίθμου(return failure)**

ii. **ELSE**
    1. ΛΠ = ΛΠ + (z,x) {για κάθε (z,x) τέτοιο ώστε να υπάρχει ΔΠ(x,z)}
iii. **ΤΕΛΟΣ_IF**

4. **ΤΕΛΟΣ_IF**

7. **WHILE** ΛΠ να είναι άδεια

<u>Συνάρτηση arc-reduce(x,y)</u>

1. Μεταβολή = **false**
2. **FOR** κάθε vx στο ΠΟ(x)
    1. Εύρεση τιμής vy στο ΠΟ(y) τέτοια ώστε vx και vy να ικανοποιούν τον περιορισμό ΔΠ(x,y)
    2. **IF** δεν υπάρχει vy
        i. ΠΟ(x) = ΠΟ(x) - vx
        ii. Μεταβολή = **true**
    3. **ΤΕΛΟΣ_IF**
3. **ΤΕΛΟΣ_FOR**
4. **ΕΠΕΣΤΡΕΨΕ** Μεταβολή

Στην αρχή του αλγορίθμου, ελέγχεται αν ικανοποιούνται οι μοναδιαίοι περιορισμοί και γίνεται διαγραφή των τιμών από τα πεδία ορισμού των μεταβλητών που δεν τους ικανοποιούν. Θα πρέπει να σημειωθεί ότι δεν είναι απαραίτητη η ύπαρξη μοναδιαίων περιορισμών στα CSPs. Στην συνέχεια προστίθενται στην λίστα περιορισμών κάθε τόξο και το αντίστροφο του. Σε κάθε επανάληψη, επιλέγεται το πρώτο τόξο (X,Y) από την λίστα περιορισμών και αφαιρείται (η λίστα λειτουργεί σαν ουρά, ακολουθείται η σειρά FIFO), όπου x και y οι μεταβλητές των περιορισμών. Ελέγχεται για κάθε τιμή της X αν υπάρχει αντίστοιχη τιμή στην Y που να επαληθεύει τον περιορισμό, αν δεν υπάρχει τότε διαγράφεται από το πεδίο ορισμού της X. Αν διαγραφτεί κάποια τιμή, προστίθενται στο τέλος της λίστας οι περιορισμοί που συνδέονται με την μεταβλητή X για επανέλεγχο, εφόσον έχει μεταβληθεί το πεδίο ορισμού της (δεν προστίθενται αντίγραφα περιορισμών στην λίστα). Όταν αδειάσει η λίστα περιορισμών οι μεταβλητές είναι arc consistent μεταξύ τους, ενώ αν αδειάσει ένα πεδίο ορισμού μιας μεταβλητής ο αλγόριθμος τερματίζει πρόωρα

Στο παρακάτω παράδειγμα του Dr John Levine, καθηγητή του πανεπιστημίου Strathclyde της Γλασκώβης, φαίνεται η μετατροπή της λίστας περιορισμών και των πεδίων ορισμού των μεταβλητών κατά την διάρκεια του αλγορίθμου.

Ας υποθέσουμε ότι έχουμε 3 μεταβλητές Α{1,2,3}, Β{1,2,3}, Γ{1,2,3} με περιορισμούς Α>Β και Β=Γ. Τα τόξα μας είναι τα Α>Β, το αντίστροφο του Β<Α, Β=Γ και το αντίστροφο του Γ=Β, τα αποθηκεύουμε προσωρινά στην λίστα περιορισμών.

**Α{1,2,3}, Β{1,2,3}, Γ{1,2,3}**

Λίστα περιορισμών: Α>Β, Β<Α, Β=Γ, Γ=Β

Ελέγχουμε το Α>Β, η τιμή 1 του πεδίου ορισμού του Α δεν ικανοποιεί τον περιορισμό οπότε διαγράφεται. Το τόξο Β<Α υπάρχει ήδη στην λίστα περιορισμών άρα δεν ανανεώνεται.

**Α{2,3}, Β{1,2,3}, Γ{1,2,3}**

Λίστα περιορισμών: Β<Α, Β=Γ, Γ=Β

Ελέγχουμε το Β<Α, η τιμή 3 του πεδίου ορισμού του Β δεν ικανοποιεί τον περιορισμό οπότε διαγράφεται. Προστίθεται το τόξο Α>Β στην λίστα περιορισμών.

**Α{2,3}, Β{1,2}, Γ{1,2,3}**

Λίστα περιορισμών: Β=Γ, Γ=Β, Α>Β

Ελέγχουμε το Β=Γ, όλες οι τιμές του Β επαληθεύουν τον περιορισμό.

**Α{2,3}, Β{1,2}, Γ{1,2,3}**

Λίστα περιορισμών: Γ=Β, Α>Β

Ελέγχουμε το Γ=Β, η τιμή 3 του πεδίου ορισμού του Γ δεν ικανοποιεί τον περιορισμό οπότε διαγράφεται. Προστίθεται το τόξο Β=Γ στην λίστα περιορισμών.

**Α{2,3}, Β{1,2}, Γ{1,2}**

Λίστα περιορισμών: Α>Β, Β=Γ

Ελέγχουμε το Α>Β, όλες οι τιμές του Α επαληθεύουν τον περιορισμό.

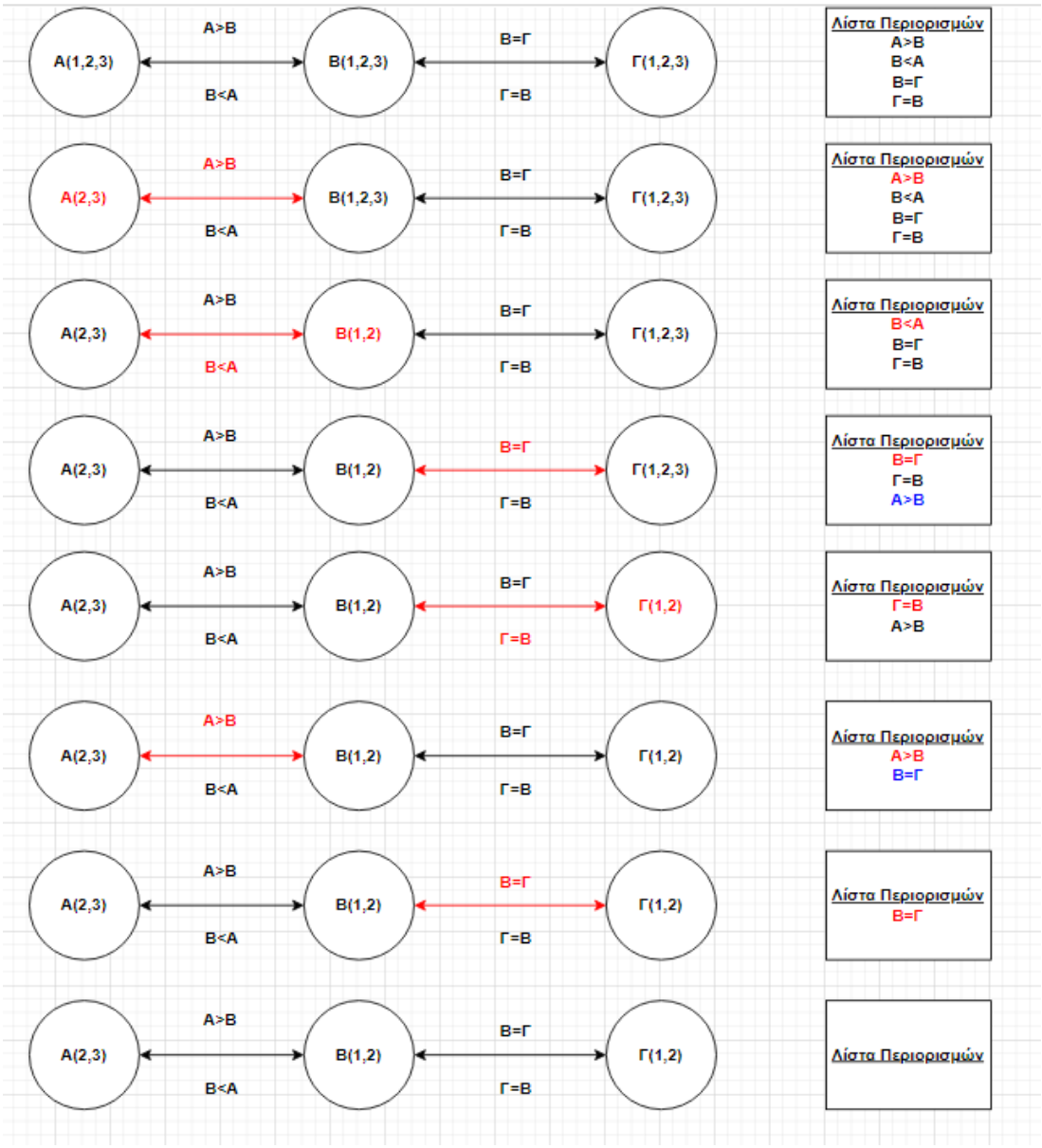
**Α{2,3}, Β{1,2}, Γ{1,2}**

Λίστα περιορισμών: Β=Γ

Ελέγχουμε το Β=Γ, όλες οι τιμές του Β επαληθεύουν τον περιορισμό.

Η λίστα περιορισμών έχει αδειάσει, όλοι οι περιορισμοί ικανοποιήθηκαν οπότε όλα τα πεδία ορισμού είναι συνεπή ως προς τα τόξα.

Στην παρακάτω εικόνα απεικονίζεται με κόκκινη γραμματοσειρά ποιος περιορισμός εξετάζεται σε κάθε βήμα και το ανανεωμένο πεδίο ορισμού της μεταβλητής, σε περίπτωση που αλλάξει. Με μπλε χρώμα φαίνονται τα τόξα που προστίθενται στην λίστα των περιορισμών.


A(1,2,3)
A>B
B<A
B(1,2,3)
B=Γ
Γ=B
Γ(1,2,3)
Λίστα Περιορισμών
A>B
B<A
B=Γ
Γ=B
A(2,3)
A>B
B<A
B(1,2,3)
B=Γ
Γ=B
Γ(1,2,3)
Λίστα Περιορισμών
A>B
B<A
B=Γ
Γ=B
A(2,3)
A>B
B<A
B(1,2)
B=Γ
Γ=B
Γ(1,2,3)
Λίστα Περιορισμών
B<A
B=Γ
Γ=B
A(2,3)
A>B
B<A
B(1,2)
B=Γ
Γ=B
Γ(1,2,3)
Λίστα Περιορισμών
B=Γ
Γ=B
A>B
A(2,3)
A>B
B<A
B(1,2)
B=Γ
Γ=B
Γ(1,2)
Λίστα Περιορισμών
Γ=B
A>B
A(2,3)
A>B
B<A
B(1,2)
B=Γ
Γ=B
Γ(1,2)
Λίστα Περιορισμών
A>B
B=Γ
A(2,3)
A>B
B<A
B(1,2)
B=Γ
Γ=B
Γ(1,2)
Λίστα Περιορισμών
B=Γ
A(2,3)
A>B
B<A
B(1,2)
B=Γ
Γ=B
Γ(1,2)
Λίστα Περιορισμών

## 3.3 Μια αναφορά στους υπόλοιπους Arc Consistency αλγορίθμους

### 3.3.1 Αλγόριθμος AC-1

Ο πρώτος αλγόριθμος συνέπειας τόξου, ο AC-1 (Arc Consistency Algorithm #1), διαχειρίζεται τις μεταβλητές (variables), τα πεδία των ορισμών τους (domains) και τους περιορισμούς μεταξύ τους (constraints) με απλό τρόπο. Βασικό του μειονέκτημα αποτελεί ότι ελέγχει κάθε φορά όλους τους περιορισμούς. Υπάρχει πιθανότητα ο αλγόριθμος να κάνει περιττούς ελέγχους, για παράδειγμα σε μια επανάληψη να διαγραφεί μια τιμή του πεδίου ορισμού μιας μεταβλητής Α και στην επομένη να ελεγχθούν ξανά οι περιορισμοί μιας μεταβλητής Γ, χωρίς να υπάρχει απαραίτητα τόξο που να συνδέει τις μεταβλητές Α και Γ **[11][12]**.

### 3.3.2 Αλγόριθμος AC-2

Ο δεύτερος αλγόριθμος συνέπειας τόξου, ο AC-2 (Arc Consistency Algorithm #2), βελτίωσε το μειονέκτημα του αλγορίθμου AC-1 με τους περιττούς ελέγχους, ελέγχοντας μετά τον πρώτο έλεγχο όλων των περιορισμών μόνο τα τόξα των οποίων άλλαξε το πεδίο ορισμού των μεταβλητών τους. Το μειονέκτημα του AC-2 είναι πιθανά θέματα μνήμης που μπορεί να προκληθούν κατά την εκτέλεση του αλγόριθμου. Μια απλοποιημένη και βελτιωμένη έκδοση του AC-2 είναι ο AC-3, ο οποίος επεξηγήθηκε στο προηγούμενο κεφάλαιο.

### 3.3.3 Αλγόριθμος AC-4

Ο αλγόριθμος AC-4 δημιουργήθηκε από τους Mohr και Henderson και βασίζεται στην έννοια της υποστήριξης, δηλαδή μια τιμή A του πεδίου ορισμού της μεταβλητής x υποστηρίζεται από μια τιμή B του πεδίου ορισμού της μεταβλητής y όταν για x=A και y=B ικανοποιείται ο περιορισμός των

μεταβλητών **[12]**. Ο αλγόριθμος χρησιμοποιεί 2 παραπάνω δομές δεδομένων, ένα counter στο οποίο μετράμε σε κάθε τόξο πόσες τιμές της μεταβλητής y υπάρχουν που υποστηρίζουν την μεταβλητή x και ένα support set στο οποίο για κάθε τιμή της μεταβλητής x ποιες είναι οι τιμές της y που την υποστηρίζουν. Όταν το counter μηδενιστεί η τιμή χαρακτηρίζεται περιττή και διαγράφεται. Ο AC-4 έχει μικρότερη πολυπλοκότητα χρόνου και μεγαλύτερη πολυπλοκότητα χώρου από τον AC-3, $O(cd^2)$ και $O(cd^2)$ αντίστοιχα, όπου c ο αριθμός των περιορισμών και d το μέγεθος του μεγαλύτερου πεδίου ορισμού **[13]**.

### 3.3.4 Αλγόριθμος AC-5

Ο αλγόριθμος AC-5 διαχειρίζεται μια ουρά με τόξα (x,y) και τις διαγραφόμενες τιμές από το πεδίο ορισμού του y, εφόσον διαγράφεται μια τιμή πρέπει να ξαναελεγχθεί το τόξο (x,y). Ο αλγόριθμος έχει δύο βασικά βήματα, στην αρχή εφαρμόζεται η συνέπεια τόξου σε όλα τα τόξα και διαγράφονται οι τιμές που δεν επαληθεύονται περιορισμοί, έπειτα γίνεται τοπική συνέπεια τόξου στα υπόλοιπα στοιχεία της ουράς το οποίο είναι πιθανό να δημιουργήσει νέα στοιχεία σε αυτή. Η πολυπλοκότητα χρόνου είναι $O(cd^2)$ και η πολυπλοκότητα χώρου O(c+nd), όπου c ο αριθμός των περιορισμών, d το μέγεθος του μεγαλύτερου πεδίου ορισμού και n ο αριθμός των μεταβλητών **[14]**.

### 3.3.5 Αλγόριθμος AC-6

Ο αλγόριθμος AC-6 **[15]** βελτιώνει τον αλγόριθμο AC-4 με έναν απλό τρόπο, δεν αποθηκεύει όλους τους αριθμούς του y που ικανοποιούν τον περιορισμό αλλά αποθηκεύει τον μικρότερο αριθμό του y που τον επαληθεύει. Αυτό προϋποθέτει να έχει προηγηθεί μια ταξινόμηση των αριθμών των πεδίων ορισμών σε αύξουσα σειρά. Η πολυπλοκότητα χρόνου του αλγορίθμου είναι $O(ad^2)$ και η πολυπλοκότητα χώρου O(ad), όπου a ο αριθμός των περιορισμών και d ο αριθμός των ζευγαριών που υποστηρίζονται.

### 3.3.6 Αλγόριθμος AC-7

Ο αλγόριθμος AC-7, σε αντίθεση με τον AC-3, AC-4 και AC-6, δεν αναζητάει την υποστήριξη δύο μεταβλητών αλλά την συμπεραίνει λαμβάνοντας υπόψιν ότι είναι αμφίδρομη, δηλαδή όταν μια τιμή A του πεδίου ορισμού της μεταβλητής x υποστηρίζεται από μια τιμή B του πεδίου ορισμού της μεταβλητής y τότε και η τιμή B υποστηρίζεται από την τιμή A. Για κάθε ζευγάρι υπάρχει μια βοηθητική σειρά που περιέχει τις τιμές της y που υποστηρίζονται από τις τιμές της x, μια τιμή last, η οποία δείχνει την τελευταία τιμή που ελέγχθηκε να υποστηρίζει την τιμή A της μεταβλητής x και βοηθητικές μεταβλητές top και bottom, υψηλότερη και χαμηλότερη τιμή αντίστοιχα από όλες τις τιμές των πεδίων ορισμών των μεταβλητών. Η πολυπλοκότητα χρόνου του αλγορίθμου είναι $O(ad^2)$ και η πολυπλοκότητα χώρου O(ad), όπου a ο αριθμός των περιορισμών και d ο αριθμός των ζευγαριών που υποστηρίζονται **[16]**.

### 3.3.7 Αλγόριθμος AC2000

Ο αλγόριθμος AC2000 βελτιώνει τον AC-3, όταν διαγραφτεί μια τιμή από το πεδίο ορισμού της y δεν ξαναελέγχεται απαραίτητα αν όλες οι τιμές της μεταβλητής x επαληθεύουν τον περιορισμό, ελέγχεται πρώτα αν η μεταβλητή x έχασε κάποια μεταβλητή στην Y που την υποστήριζε. Για να γίνει αυτό χρησιμοποιείται μια βοηθητική δομή δεδομένων, στην οποία αποθηκεύονται οι τιμές της y που διαγράφτηκαν. Αυτή η αναζήτηση υποστήριξης ονομάζεται “lazymode”, χρησιμοποιείται όταν το τωρινό πεδίο ορισμού της y είναι αρκετά μικρότερο από το αρχικό της λαμβάνοντας υπόψιν μια βοηθητική παράμετρο Ratio. Η πολυπλοκότητα χρόνου του αλγορίθμου είναι $O(ad^3)$ και η πολυπλοκότητα χώρου O(ad), όπου το a είναι ο αριθμός των τόξων και d είναι το μέγεθος του μεγαλύτερου πεδίου ορισμού **[17]**.

### 3.3.8 Αλγόριθμος AC2001

Ο αλγόριθμος AC2001 βελτιώνει τον AC2000 αποθηκεύοντας την τελευταία τιμή των μεταβλητών που υποστηρίζουν τους περιορισμούς. Εφόσον είναι γνωστό ποια είναι η τελευταία τιμή, ελέγχεται αν έχει αφαιρεθεί αυτή χωρίς να χρειάζεται να ελεγχθεί όλο το πεδίο ορισμού της μεταβλητής y, αν έχει αφαιρεθεί ελέγχονται οι τιμές που είναι μετά από αυτή. Ο αλγόριθμος χρησιμοποιεί συνέχεια το “lazymode”, δεν υπάρχει η παράμετρος Ratio. Η πολυπλοκότητα χρόνου του αλγορίθμου είναι $O(ad^2)$ και η πολυπλοκότητα χώρου O(ad), όπου το a είναι ο αριθμός των τόξων και d είναι το μέγεθος του μεγαλύτερου πεδίου ορισμού. Ο αλγόριθμος AC2001 είναι ο πρώτος optimized arc consistent αλγόριθμος που δεν χρησιμοποιεί λίστες με τιμές μεταβλητών που υποστηρίζονται **[17]**.

Γενικότερα, λόγω της απλότητας της υλοποίησης του συγκριτικά με τους μεταγενέστερους αλγορίθμους και της καλής πολυπλοκότητας του, ο AC-3 είναι ένας αλγόριθμος που προτιμάται για την επίλυση των CSPs. Πρέπει να σημειωθεί ότι ο Arc Consistency δεν εγγυάται ότι αν ένα CSP είναι συνεπές θα έχει λύση ή ποια θα είναι αυτή. Υπάρχουν φορές που μπορεί να βρει λύση, συμπεραίνεται πρόωρα ότι αν αδειάσει το πεδίο ορισμού μιας μεταβλητής δεν υπάρχει λύση, αλλά η τυπική χρήση της συνέπειας τόξου είναι να μικρύνει τον χώρο αναζήτησης του CSP.

# Κεφάλαιο 4

# Simulated Annealing

## 4.1 Εισαγωγή στον Simulated Annealing

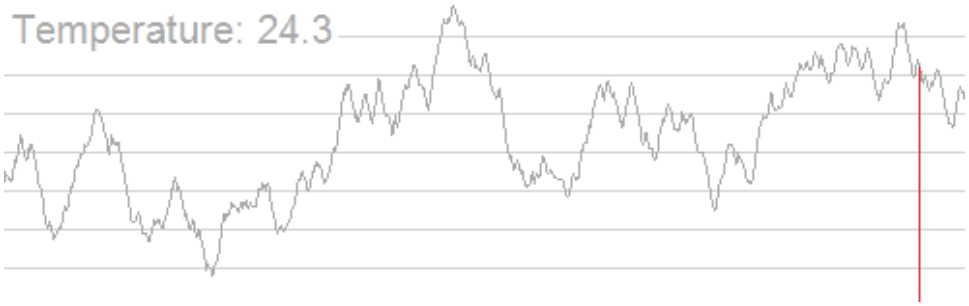
Ο Simulated Annealing (SA) είναι μια μέθοδος επίλυσης προβλημάτων βελτιστοποίησης, είτε με περιορισμούς είτε χωρίς. Πιο συγκεκριμένα, είναι ένα metaheuristic που επιχειρεί να προσεγγίσει την καλύτερη δυνατή βελτιστοποίηση σε ένα εκτενή χώρο αναζήτησης των προβλημάτων βελτιστοποίησης. Metaheuristic χαρακτηρίζονται τα υψηλότερου επιπέδου heuristic που είναι σχεδιασμένα να βρουν, να δημιουργήσουν, να τροποποιήσουν ή να διαλέξουν ένα heuristic, τα οποία μπορούν να παρέχουν μια ικανοποιητική λύση για προβλήματα βελτιστοποίησης **[18].**

Η συγκεκριμένη μέθοδος εφαρμόζεται σε προβλήματα ικανοποίησης περιορισμών αναθέτοντας τιμές στις μεταβλητές με κριτήριο πόσοι περιορισμοί παραβιάζονται. Αν μετά από κάποιο χρονικό διάστημα δεν παραβιάζεται κάποιος περιορισμός, τότε η συγκεκριμένη ανάθεση αποτελεί λύση του CSP. Η τεχνική πήρε το όνομα της από τους Kirkpatrick, Gelatt Jr. και Vecchi το 1983 και προτάθηκε ως λύση στο πρόβλημα του πλανόδιου πωλητή (Traveling Salesman Problem, TSP) **[19]**.

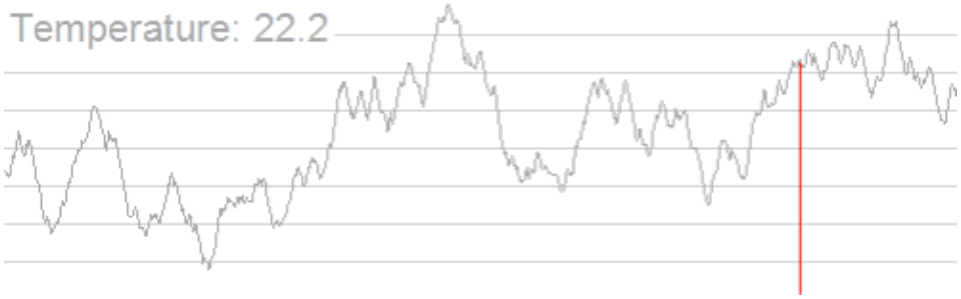
Το όνομα του Simulated Annealing προέρχεται από την διαδικασία της ανόπτησης (Annealing) στην μεταλλουργία. Η ανόπτηση είναι η θερμική επεξεργασία κατά την οποία μεταβάλλεται η φυσική και χημική ιδιότητα του μετάλλου με αποτέλεσμα να παίρνει διάφορες μορφές από την αρχική του. Όταν το μέταλλο “κρυώσει” έχει πάρει μια νέα μορφή και έχει διαφορετικές χρήσεις, συγκριτικά με την αρχική του. Παρόμοια με την ανόπτηση του μετάλλου, όταν η θερμοκρασία είναι υψηλή ο Simulated Annealing αποδέχεται μεταβολές οι οποίες δεν είναι απαραίτητα βέλτιστες, ενώ όταν η θερμοκρασία είναι χαμηλή επιλέγει κυρίως τιμές που βελτιώνουν την κατάσταση στην οποία βρίσκεται. Η προσωρινή αποδοχή μη βέλτιστων τιμών επιτρέπει στην αναζήτηση να είναι πιο ευέλικτη, εκτεταμένη και να μην

σταματάει σε τοπικά μέγιστα αλλά να αναζητεί το ολικό μέγιστο. Στις παρακάτω εικόνες φαίνεται πως στην αρχή (Temperature: 25.0) η αναζήτηση του ολικού μέγιστου είναι μακριά από το ολικό μέγιστο και πως σταδιακά πλησιάζει σε αυτό **[20]**.

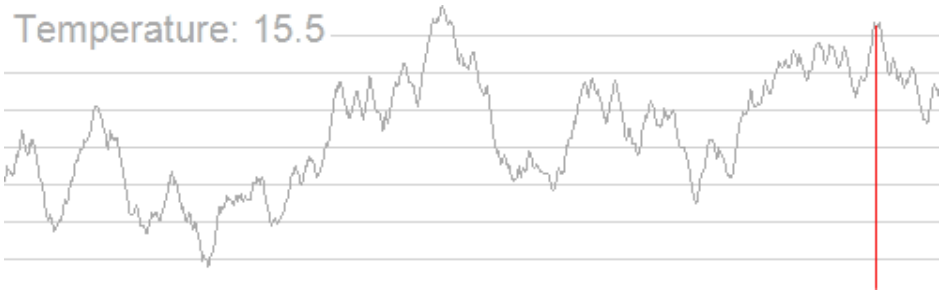
Θερμοκρασία: 24.3



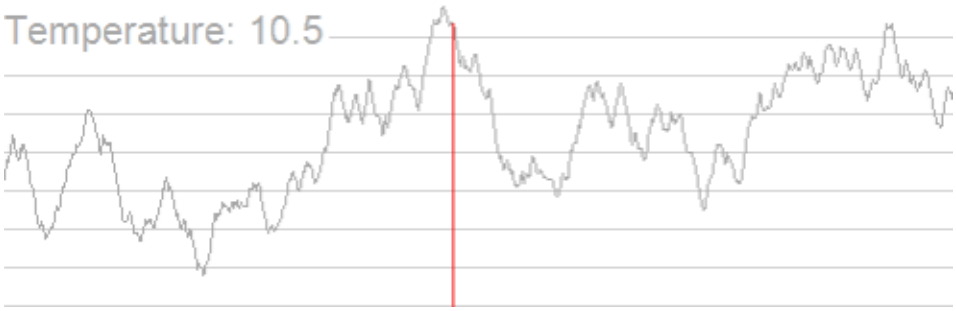

Θερμοκρασία: 22.2



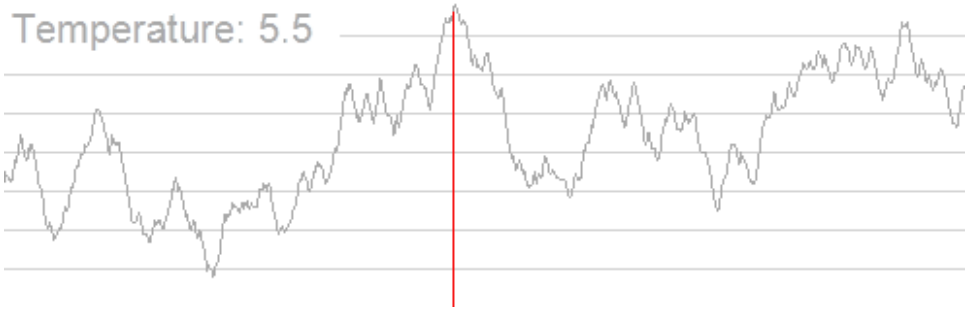

Θερμοκρασία: 15.5



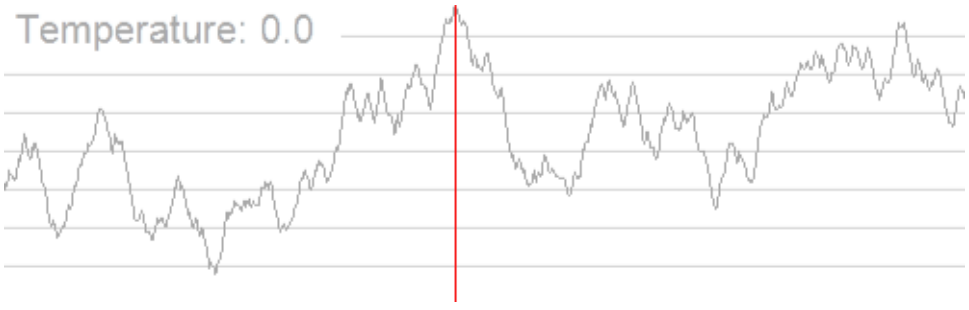
Θερμοκρασία: 10.5




Θερμοκρασία: 5.5




Θερμοκρασία: 0.0



## 4.2 Αλγόριθμος Simulated Annealing

Παρακάτω παρουσιάζεται ο ψευδοκώδικας του αλγορίθμου Simulated Annealing

### Simulated Annealing

### Είσοδος

Αρχική τυχαία ανάθεση $s_0$, $T_{αρχική}$, a

### Έξοδος

Καλύτερη ανάθεση που έχει βρεθεί όταν η θερμοκρασία T φτάσει την θερμοκρασία $T_{min}$

1. s <-- $s_0$
2. $T_{προσωρινή}$ <-- $T_{αρχική}$
3. **WHILE** $T_{προσωρινή}$ > $T_{min}$
    1. **FOR** i ΑΠΟ 1 MEXPI max_iterations
        1. $s_{next}$ = s
        2. $s_{next}$ = PerturbRandomVariable(s)
        3. Δ <-- cost($s_{next}$) - cost(s)
        4. **IF** Δ ≤ 0
            i. s <-- $s_{next}$
        5. **ELSE IF** ρ > random
            i. s <-- $s_{next}$
    2. **ΤΕΛΟΣ_FOR**
    3. $T_{προσωρινή}$ = $T_{προσωρινή}$ * a
4. **ΤΕΛΟΣ_WHILE**
5. **ΕΠΕΣΤΡΕΨΕ** s

Για την επίλυση προβλημάτων ικανοποίησης περιορισμών, αρχικά δημιουργείται μια πλήρης ανάθεση όπου κάθε μεταβλητή παίρνει και μια τιμή από το πεδίο ορισμού της. Η ανάθεση αυτή ονομάζεται $s_0$. Αρχικοποιείται ως $T_{προσωρινή}$ η τιμή της $T_{αρχική}$, η οποία εισάγεται από τον χρήστη.

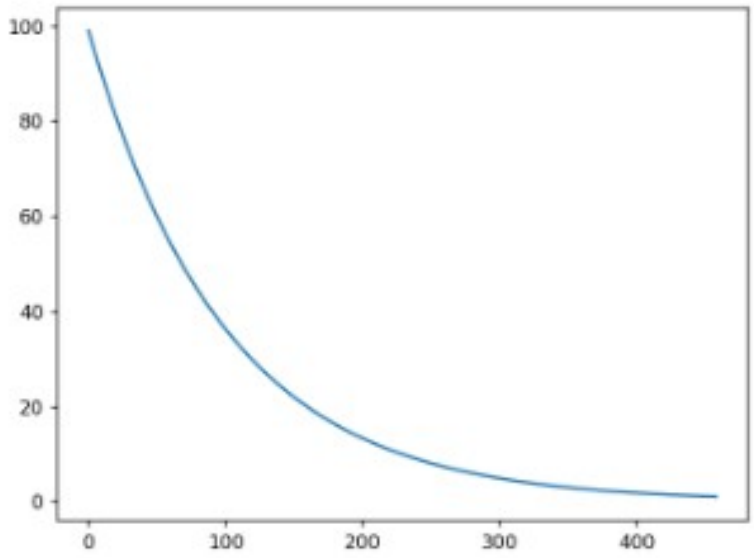
Οι τιμές των μεταβλητών max_iterations (αριθμός επαναλήψεων) και συντελεστής ψύξης θερμοκρασίας a (Temperature cooling coefficient, a<1) εισάγονται και αυτές από τον χρήστη. Όσο η προσωρινή θερμοκρασία είναι μικρότερη από την ελάχιστη $T_{min}$ θερμοκρασία που έχουμε θέσει (συνήθως παίρνει χαμηλές τιμές π.χ. ≤1) τότε επαναλαμβάνεται ο εξής βρόγχος: για max_iterations επαναλήψεις δημιουργείται μια γειτονική κατάσταση $s_{next}$ . Η $s_{next}$ διαφέρει από την s στην τιμή μιας τυχαίας μεταβλητής. Υπολογίζεται και για τις δύο το κόστος τους, δηλαδή πόσοι περιορισμοί παραβιάζονται. Αν η διαφορά κόστους Δ είναι μικρότερη από 0, τότε γνωρίζουμε ότι η $s_{next}$ είναι καλύτερη ανάθεση γιατί παραβιάζονται λιγότεροι περιορισμοί, επομένως αποθηκεύεται στην s και συνεχίζεται με αυτή στις επόμενες επαναλήψεις. Αν η $s_{next}$ είναι χειρότερη σαν ανάθεση, υπάρχει πιθανότητα να γίνει δεκτή αν η πιθανότητα ρ είναι μεγαλύτερη από την τιμή random, όπου random μια τυχαία τιμή στο κλειστό διάστημα [0,1]. Η τιμή ρ ορίζεται ως:

$$\rho = e^{-\Delta/T_{προσωρινή}}$$

όπου Δ η διαφορά του κόστους μεταξύ $s_{next}$ και s.

Στα αρχικά στάδια του αλγορίθμου, η θερμοκρασία είναι υψηλή οπότε υπάρχει μεγάλη πιθανότητα να επαληθεύεται η σχέση ρ > random και να γίνει δεκτή μια μετάβαση σε γειτονική κατάσταση που είναι χειρότερη από την τρέχουσα. Όπως αναφέρθηκε και προηγουμένως, η αποδοχή κινήσεων προς χειρότερες καταστάσεις επιτρέπει στον Simulated Annealing να είναι πιο ευέλικτος στην αναζήτηση του και να μην σταματάει σε τοπικά μέγιστα ή ελάχιστα αντίστοιχα, αναλόγως με το πρόβλημα που καλείται να επιλύσει. Στο τέλος του εξωτερικού βρόγχου, η προσωρινή θερμοκρασία μειώνεται με βάση τον συντελεστή ψύξης θερμοκρασίας a. Όσο μειώνεται η θερμοκρασία, η πιθανότητα ρ τείνει προς μικρότερες τιμές, οπότε είναι λιγότερο πιθανό να γίνει αποδεκτή μια μετάβαση προς χειρότερη κατάσταση. Για παράδειγμα, αν

$\Delta = 20$ και $T_{προσωρινή} = 100$, τότε η πιθανότητα ρ θα ισούται με ρ = 0.818, οπότε έχει υψηλή πιθανότητα αποδοχής. Όταν η $T_{προσωρινή}$ θα ισούται με $T_{προσωρινή} = 10$ και τύχει η διαφορά κόστους να ξαναισούται με $\Delta = 20$, τότε η πιθανότητα ρ θα ισούται με ρ = 0.135, οπότε έχει αρκετά μικρότερη πιθανότητα αποδοχής **[21]**.




Στο παρακάτω διάγραμμα απεικονίζεται η μείωση της θερμοκρασίας $T_{αρχική} = 100$ με τον συντελεστή a=0.99.

# Κεφάλαιο 5

# Υβριδικοί Αλγόριθμοι

## 5.1 Εισαγωγή στους Υβριδικούς Αλγορίθμους

Στο Κεφάλαιο 3 μελετήθηκε η συνέπεια τόξου και οι αλγόριθμοι της και στο Κεφάλαιο 4 ο Simulated Annealing και ο ψευδοκώδικας του. Η συνέπεια τόξου είναι ένα εργαλείο διάδοσης περιορισμών και βοηθάει στην μείωση των χώρων αναζήτησης των CSPs. Ο Simulated Annealing είναι μια μέθοδος επίλυσης CSPs και δημιουργεί μια ανάθεση τιμών, η οποία μπορεί να είναι ικανοποιητική για το πρόβλημα που προσπαθεί να επιλύσει. Στο συγκεκριμένο κεφάλαιο θα αναλυθούν οι υβριδικοί αλγόριθμοι μεταξύ των δύο μεθόδων, και πιο συγκεκριμένα το πως η συνέπεια τόξου μπορεί να επηρεάσει τον Simulated Annealing στην δημιουργία ανάθεσης τιμών με σκοπό την επίλυση των προβλημάτων.

## 5.2 Μέθοδος 1: Μειωμένος Χώρος Αναζήτησης

Ο πιο απλός συνδυασμός των δύο μεθόδων είναι με την χρήση του μειωμένου χώρου αναζήτησης. Στο πρόβλημα ικανοποίησης περιορισμών που καλείται να επιλυθεί χρησιμοποιείται πρώτα η συνέπεια τόξου. Αν το πρόβλημα είναι συνεπές τότε χρησιμοποιείται ο Simulated Annealing με τα ανανεωμένα πεδία ορισμού των μεταβλητών. Στην συγκεκριμένη μέθοδο η συνέπεια τόξου επηρεάζει έμμεσα την επιλογή του Simulated Annealing, συγκριτικά με την επόμενη μέθοδο.

Στην αρχή καλείται η συνέπεια τόξου. Αν το πρόβλημα είναι ασυνεπές τότε σταματάει η αναζήτηση λύσης. Αν είναι συνεπές, καλείται ο Simulated

Annealing με το νέο ανανεωμένο πεδίο ορισμού. Επιστρέφεται η λύση του προβλήματος, η οποία αν είναι συνεπής και ολοκληρωμένη γίνεται αποδεκτή.

Η συγκεκριμένη μέθοδος στοχεύει στην απαλοιφή τιμών μέσω της συνέπειας τόξου και στην "σωστότερη" επιλογή τιμών του Simulated Annealing, σε σύγκριση με την απλή μέθοδο.

## 5.3 Μέθοδος 2: Επιρροή της συνέπειας τόξου στην επιλογή τιμής

Η συνέπεια τόξου στην συγκεκριμένη μέθοδο επηρεάζει άμεσα την επιλογή τιμής στον Simulated Annealing. Στο πρόβλημα ικανοποίησης περιορισμών που καλείται να επιλυθεί χρησιμοποιείται αρχικά ο Simulated Annealing. Σε κάθε επανάληψη ο αλγόριθμος επιλέγει μια τυχαία μεταβλητή στην ανάθεση s_next(αντίγραφο της s) για να αλλάξει την τιμή της. Στην απλή μέθοδο του SA η τιμή της επιλέγεται τυχαία από το πεδίο ορισμού της. Στην συγκεκριμένη μέθοδο επεμβαίνει η συνέπεια τόξου. Επιλέγεται τυχαία η μεταβλητή, έπειτα για κάθε τιμή της χρησιμοποιείται μια περιορισμένη μορφή της συνέπειας τόξου. Η τιμή που παραβιάζει τους λιγότερους περιορισμούς είναι η "καλύτερη" επιλογή, οπότε επιλέγεται αυτή ως τιμή της μεταβλητής στην s_next.

Παρακάτω εμφανίζεται ο ψευδοκώδικας της συγκεκριμένης μεθόδου.

Συνάρτηση Simulated Annealing (Αρχικό Πεδίο Ορισμού, Μεταβλητές, Περιορισμοί)

1. s <-- $s_0$
2. $T_{προσωρινή}$ <-- $T_{αρχική}$
3. **WHILE** $T_{προσωρινή} > T_{min}$
    1. **FOR** i ΑΠΟ 1 MEXPI max_iterations
        1. $s_{next}$ = s

2. Επιλογή τυχαίας μεταβλητής στην $s_{next}$
3. **FOR** κάθε τιμή της μεταβλητής που επιλέχθηκε
   i. ΛΠ (Λίστα Περιορισμών) = Άδεια
   ii. **FOR** κάθε Δυαδικό Περιορισμό (x,y)
      i. ΛΠ = ΛΠ + (x,y) + (y,x)
   iii. **ΤΕΛΟΣ_FOR**
   iv. Arc Consistency (Περιορισμένο Πεδίο Ορισμού, Μεταβλητές, ΛΠ, Παραβιάσεις)
4. **ΤΕΛΟΣ_FOR**
5. Τιμή μεταβλητής στην $s_{next}$=FindMin(Παραβιάσεις, ΠΟ)
6. Δ <-- cost($s_{next}$) - cost(s)
7. **IF** Δ ≤ 0
   i. s <-- $s_{next}$
8. **ELSE** IF ρ > random
   i. s <-- $s_{next}$
9. **ΤΕΛΟΣ_IF**

2. **ΤΕΛΟΣ_FOR**

4. $T_{προσωρινή} = T_{προσωρινή} * a$
5. **ΤΕΛΟΣ_WHILE**
6. **ΕΠΕΣΤΡΕΨΕ** s

<u>Συνάρτηση Arc Consistency (Περιορισμένο Πεδίο Ορισμού, Μεταβλητές, ΛΠ, Παραβιάσεις)</u>

1. **DO**
   1. Επέλεξε ένα τόξο (x,y) από ΛΠ
   2. ΛΠ := ΛΠ - (x,y)
   3. **IF arc-reduce(x,y) = true**{
      i. **IF** ΠΔ(x) είναι άδειο
         1. **Τερματισμός Αλγορίθμου(return failure)**
      ii. **ΤΕΛΟΣ_IF**
   4. **ELSE**
      i. ΛΠ := ΛΠ + (z,x) + (x,z)
   5. **ΤΕΛΟΣ_IF**
2. **WHILE** ΛΠ να είναι άδεια

## Συνάρτηση arc-reduce (x,y)

1. Μεταβολή = **false**
    1. **FOR** κάθε vx στο ΠΟ(x)
        1. Εύρεση τιμής vy στο ΠΟ(y) τέτοια ώστε vx και vy να ικανοποιούν τον περιορισμό ΔΠ(x,y)
        2. **IF** δεν υπάρχει vy
            i. ΠΟ(x) = ΠΟ(x) - vx
            ii. Μεταβολή = **true**
        3. **ΤΕΛΟΣ_IF**
    2. **ΤΕΛΟΣ_FOR**
2. **ΕΠΕΣΤΡΕΨΕ** Μεταβολή

## Συνάρτηση FindMin (Παραβιάσεις, ΠΟ)

1. ΕλάχιστηΤιμή = Πρώτη Θέση στον Πίνακα Παραβιάσεων
2. **FOR** κάθε τιμή του ΠΟ
    1. **IF** ΤωρινήΘέση(στον Πίνακα Παραβιάσεων)<ΕλάχιστηΤιμή
        i. ΕλάχιστηΤιμή = ΤωρινήΘέση
    2. **ΤΕΛΟΣ_IF**
    3. **IF** υπάρχουν 2 ή παραπάνω τιμές με ίδια ΕλάχιστηΤιμή
        i. Τυχαία Επιλογή Τιμής από το ΠΟ με τις λιγότερες παραβιάσεις
    4. **ΤΕΛΟΣ_IF**
3. **ΤΕΛΟΣ_FOR**
4. **ΕΠΕΣΤΡΕΨΕ** Τιμή από το ΠΟ

## CSP (Αρχικό Πεδίο Ορισμού, Μεταβλητές, Περιορισμοί)

1. **Κάλεσε Simulated Annealing** (Αρχικό Πεδίο Ορισμού, Μεταβλητές, Περιορισμοί)
2. **ΕΜΦΑΝΙΣΕ** s

Κάθε φορά που καλείται η συνάρτηση της συνέπειας τόξου, το Περιορισμένο Πεδίο Ορισμού είναι ίδιο με το Αρχικό Πεδίο Ορισμού εκτός από τις τιμές της τυχαίας μεταβλητής που επιλέχθηκε. Σε κάθε επανάληψη το πεδίο ορισμού της μεταβλητής περιέχει διαφορετική τιμή από το πεδίο ορισμού της και μόνο αυτή. Για παράδειγμα, αν η μεταβλητή C έχει τις τιμές {a,b,c}, στην πρώτη επανάληψη στο Περιορισμένο Πεδίο Ορισμού θα έχει μόνο την τιμή a, στην επόμενη επανάληψη μόνο την τιμή b και στην τελευταία μόνο την τιμή c. Ο πίνακας Παραβιάσεις χρησιμοποιείται στην συνέπεια τόξου και μετράει τις παραβιάσεις των περιορισμών σε κάθε επανάληψη. Η βοηθητική συνάρτηση FindMin επιστρέφει την τιμή με τις λιγότερες παραβιάσεις, η οποία επιλέγεται ως τιμή της μεταβλητής. Αν δύο ή παραπάνω τιμές έχουν τον ίδιο αριθμό παραβιάσεων περιορισμών, ο αλγόριθμος επιλέγει τυχαία μια τιμή από αυτές.

Η συγκεκριμένη μέθοδος στοχεύει στην άμεση επιρροή της συνέπειας τόξου στην επιλογή των τιμών. Συγκριτικά με την απλή μέθοδο του Simulated Annealing, στην συγκεκριμένη σχεδόν πάντα επιλέγεται η καλύτερη τιμή για την μεταβλητή οπότε έχουμε optimized αναθέσεις τιμών από νωρίς στις μεταβλητές.

## 5.4 Μέθοδος 3: Επιρροή των συνολικών διαγραφών στην πιθανότητα αποδοχής τιμών

Η συγκεκριμένη συνδυαστική μέθοδος παρουσιάζει αρκετές ομοιότητες με την Μέθοδο 2. Σε κάθε επανάληψη ο αλγόριθμος επιλέγει μια τυχαία μεταβλητή στην ανάθεση s_next (αντίγραφο της s) για να αλλάξει την τιμή της. Η νέα τιμή επιλέγεται τυχαία από το πεδίο ορισμού της μεταβλητής. Έπειτα, χρησιμοποιείται μια περιορισμένη μορφή της συνέπειας τόξου για τις δύο τιμές της μεταβλητής, για την “παλιά” τιμή στην s και την “νέα” τιμή στην s_next. Η συνέπεια τόξου στην συγκεκριμένη μέθοδο υπολογίζει πόσες τιμές διαγράφονται με τις αναθέσεις και επηρεάζει την πιθανότητα αποδοχής τιμών. Παρακάτω παρουσιάζεται ο ψευδοκώδικας της συγκεκριμένης μεθόδου.

## Συνάρτηση Simulated Annealing (Αρχικό Πεδίο Ορισμού, Μεταβλητές, Περιορισμοί, Πλήθος Πιθανών Διαγραφών)

1. s <-- $s_0$, $T_{προσωρινή}$ <-- $T_{αρχική}$
2. **WHILE** $T_{προσωρινή}$ > $T_{min}$
    1. **FOR** i ΑΠΟ 1 ΜΕΧΡΙ max_iterations
        1. $s_{next}$ = s
        2. ΛΠ (Λίστα Περιορισμών) = Άδεια
        3. **FOR** κάθε Δυαδικό Περιορισμό (x,y)
            i. ΛΠ = ΛΠ + (x,y) + (y,x)
        4. **ΤΕΛΟΣ_FOR**
        5. Arc Consistency (ΠεριορισμένοΠεδίοΟρισμού 1, Μεταβλητές, ΛΠ, Πλήθος Διαγραφόμενων Τιμών)
        6. **IF** Arc Consistency τερματίσει πρόωρα
            i. Πιθανότητα_s = 1
        7. **ELSE**
            i. Πιθανότητα_s = Πλήθος Διαγραφόμενων Τιμών/ Πλήθος Πιθανών Διαγραφών
        8. **ΤΕΛΟΣ_IF**
        9. $s_{next}$ = PerturbRandomVariable(s)
        10. ΛΠ (Λίστα Περιορισμών) = Άδεια
        11. **FOR** κάθε Δυαδικό Περιορισμό (x,y)
            i. ΛΠ = ΛΠ + (x,y) + (y,x)
        12. **ΤΕΛΟΣ_FOR**
        13. Arc Consistency (ΠεριορισμένοΠεδίοΟρισμού2, Μεταβλητές,ΛΠ,Πλήθος Διαγραφόμενων Τιμών)
        14. **IF** Arc Consistency τερματίσει πρόωρα
            i. Πιθανότητα_s_next = 1
        15. **ELSE**
            i. Πιθανότητα_s _next = Πλήθος Διαγραφόμενων Τιμών/ Πλήθος Πιθανών Διαγραφών
        16. **ΤΕΛΟΣ_IF**
        17. Δ <-- [cost($s_{next}$) - cost(s)] + (Πιθανότητα_s_next- Πιθανότητα_s)
        18. **IF** Δ ≤ 0
            i. s <-- $s_{next}$
        19. **ELSE IF** ρ > random
            i. s <-- $s_{next}$

20. ΤΕΛΟΣ_IF
2. ΤΕΛΟΣ_FOR
3. $T_{προσωρινή} = T_{προσωρινή} * a$
4. ΤΕΛΟΣ_WHILE
5. ΕΠΕΣΤΡΕΨΕ s

Συνάρτηση Arc Consistency (Περιορισμένο Πεδίο Ορισμού, Μεταβλητές, ΛΠ, Πλήθος Διαγραφόμενων Τιμών)

1. DO
    1. Επέλεξε ένα τόξο (x,y) από ΛΠ
    2. ΛΠ := ΛΠ - (x,y)
    3. IF arc-reduce(x,y,Πλήθος Διαγραφόμενων Τιμών) = true
        i. IF ΠΔ(x) είναι άδειο
            i. Τερματισμός Αλγορίθμου(return failure)
        ii. ELSE
            i. ΛΠ := ΛΠ + (z,x) + (x,z)
        iii. ΤΕΛΟΣ_IF
    4. ΤΕΛΟΣ_IF
2. WHILE ΛΠ να είναι άδεια

Συνάρτηση arc-reduce (x,y,Πλήθος Διαγραφόμενων Τιμών)

1. Μεταβολή = false
2. FOR κάθε vx στο ΠΟ(x)
    1. Εύρεση τιμής vy στο ΠΟ(y) τέτοια ώστε vx και vy να ικανοποιούν τον περιορισμό ΔΠ(x,y)
    2. IF δεν υπάρχει vy{
        i. ΠΟ(x) = ΠΟ(x) – vx
        ii. Πλήθος Διαγραφόμενων Τιμών = Πλήθος Διαγραφόμενων Τιμών + 1
        iii. Μεταβολή = true
    3. ΤΕΛΟΣ_IF
3. ΤΕΛΟΣ_FOR
4. ΕΠΕΣΤΡΕΨΕ Μεταβολή

## CSP (Αρχικό Πεδίο Ορισμού, Μεταβλητές, Περιορισμοί)

1. Υπολογισμός Πλήθους Πιθανών Διαγραφών
2. **Κάλεσε Simulated Annealing** (Αρχικό Πεδίο Ορισμού, Μεταβλητές, Περιορισμοί, Πλήθος Πιθανών Διαγραφών)
3. **ΕΜΦΑΝΙΣΕ** s

Τα Περιορισμένα Πεδία Ορισμού 1 και 2 αποτελούνται από τα αρχικά πεδία ορισμών όλων των μεταβλητών εκτός από της μεταβλητής που επιλέχθηκε τυχαία, τα οποία αποτελούνται από την "παλιά" και "νέα" τιμή αντίστοιχα. Για την ανάθεση s και s_next υπολογίζεται το ποσοστό των διαγραφόμενων τιμών, το οποίο λαμβάνεται υπόψιν στην διαφορά κόστους Δ και επηρεάζει την πιθανότητα επιλογής τιμών. Η διαφορά Δ υπολογίζεται ως:

$\Delta$ <-- [cost($s_{next}$) - cost(s)] + (Πιθανότητα_s_next- Πιθανότητα_s)

Το Πλήθος Πιθανών Διαγράφων υπολογίζεται ως:

Πλήθος Πιθανών Διαγραφών = sum – varnumber

Όπου sum το σύνολο των τιμών στα πεδία ορισμού και varnumber το σύνολο των μεταβλητών. Ο λόγος αυτός υπολογίζει τον μέγιστο αριθμό των πιθανών διαγραφών όταν η κάθε μεταβλητή θα έχει μόνο μια τιμή στο πεδίο ορισμού της.

# Κεφάλαιο 6

# Πειραματικά Αποτελέσματα

## 6.1 Αποτελέσματα Graph Coloring προβλημάτων

Παρακάτω παρουσιάζονται τα αποτελέσματα των Graph Coloring προβλημάτων στα οποία χρησιμοποιήθηκε ο AC3 για να ελεγχθούν αν είναι arc consistent και ο Simulated Annealing για την ανάθεση τιμών. Οι περιορισμοί είναι της μορφής X≠Y.

Τα αποτελέσματα της τρίτης στήλης είναι με χαμηλή αρχική και τελική θερμοκρασία ($T_{αρχική}$ = 0.2, $T_{min}$ = 0.0001, a=0.9998). Με αυτές τις χαμηλές τιμές ο Simulated Annealing επιλέγει πιο σπάνια λάθος τιμή, οπότε ελέγχεται με σιγουριά αν υπάρχει λύση στο πρόβλημα (Τελικό Κόστος Small = 0) ή αν δεν υπάρχει, ποιο είναι το τελικό κόστος του προβλήματος.

Τα αποτελέσματα της τέταρτης στήλης είναι με υψηλή αρχική και σχετικά χαμηλή τελική θερμοκρασία ($T_{αρχική}$ = 100, $T_{min}$ = 1, a=0.99). Σκοπός αυτής της αναζήτησης είναι να δούμε πόσο κοντά φτάνει στην ανάθεση που βρήκε η πρώτη αναζήτηση και ποια είναι η διαφορά μεταξύ τους.

| | Arc Consistent | Κόστος Small | Κόστος Big | Διαφορά |
|---|---|---|---|---|
| **SGB Αρχεία** | | | | |
| **anna-5** | ✓ | 15 | 44 | 29 |
| **anna-8** | ✓ | 3 | 26 | 23 |
| **david-5** | ✓ | 20 | 41 | 21 |
| **david-8** | ✓ | 4 | 21 | 17 |
| **homer-5** | ✓ | 69 | 174 | 105 |
| **homer-8** | ✓ | 15 | 87 | 72 |
| **homer-10** | ✓ | 4 | 64 | 60 |

| | | | | |
|---|---|---|---|---|
| **huck-5** | ✓ | 21 | 34 | 13 |
| **huck-8** | ✓ | 6 | 14 | 8 |
| **jean-5** | ✓ | 14 | 24 | 10 |
| **jean-7** | ✓ | 4 | 15 | 11 |
| **games 120-5** | ✓ | 26 | 63 | 37 |
| **games 120-7** | ✓ | 8 | 38 | 30 |
| **games 120-8** | ✓ | 2 | 26 | 24 |
| **games 120-9** | ✓ | 0 | 30 | 30 |
| **miles250-6** | ✓ | 4 | 29 | 25 |
| **miles250-7** | ✓ | 1 | 20 | 19 |
| **miles250-8** | ✓ | 0 | 17 | 17 |
| **miles500-5** | ✓ | 102 | 155 | 53 |
| **miles500-10** | ✓ | 18 | 67 | 49 |
| **miles500-15** | ✓ | 5 | 32 | 27 |
| **miles500-18** | ✓ | 2 | 26 | 24 |
| **miles750-5** | ✓ | 243 | 305 | 62 |
| **miles750-10** | ✓ | 73 | 133 | 60 |
| **miles750-15** | ✓ | 28 | 80 | 52 |
| **miles750-20** | ✓ | 13 | 52 | 39 |
| **miles1000-5** | ✓ | 410 | 457 | 47 |
| **miles1000-10** | ✓ | 142 | 227 | 85 |
| **miles1000-15** | ✓ | 67 | 132 | 65 |
| **miles1000-20** | ✓ | 36 | 93 | 57 |
| **miles1000-25** | ✓ | 21 | 73 | 37 |
| **queen5-5-4** | ✓ | 12 | 20 | 8 |
| **queen6-6-6** | ✓ | 4 | 20 | 16 |
| **queen7-7-6** | ✓ | 16 | 37 | 21 |
| **queen8-8-8** | ✓ | 2 | 42 | 40 |
| **queen8-12-8** | ✓ | 37 | 84 | 47 |
| **queen9-9-8** | ✓ | 20 | 64 | 44 |
| **queen10-10-8** | ✓ | 48 | 89 | 41 |
| **queen10-10-12** | ✓ | 0 | 51 | 51 |
| **queen11-11-8** | ✓ | 84 | 140 | 56 |
| **queen11-11-12** | ✓ | 0 | 75 | 75 |
| **queen12-12-10** | ✓ | 56 | 133 | 77 |
| **queen12-12-13** | ✓ | 2 | 93 | 91 |
| **queen13-13-10** | ✓ | 96 | 186 | 90 |
| **queen13-13-14** | ✓ | 3 | 113 | 110 |
| **queen14-14-12** | ✓ | 64 | 186 | 122 |

| | | | | |
|---|---|---|---|---|
| **queen14-14-15** | ✓ | 4 | 128 | 124 |
| **queen15-15-13** | ✓ | 69 | 209 | 140 |
| **queen15-15-16** | ✓ | 5 | 147 | 142 |
| **queen16-16-14** | ✓ | 74 | 241 | 167 |
| **queen16-16-17** | ✓ | 4 | 170 | 166 |
| | | | | |
| **Full-Insertion** | | | | |
| **1-fullins-3-3** | ✓ | 2 | 11 | 9 |
| **1-fullins-4-4** | ✓ | 2 | 12 | 10 |
| **2-fullins-3-4** | ✓ | 1 | 10 | 9 |
| **3-fullins-3-5** | ✓ | 1 | 14 | 13 |
| **4-fullins-3-6** | ✓ | 1 | 14 | 13 |
| **5-fullins3-6** | ✓ | 3 | 32 | 29 |
| | | | | |
| **k-Insertion** | | | | |
| **1-insertions-4-3** | ✓ | 6 | 23 | 17 |
| **2-insertions-3-3** | ✓ | 1 | 7 | |
| **2-insertions-4-3** | ✓ | 8 | 30 | 22 |
| **3-insertions-3-3** | ✓ | 1 | 10 | 9 |
| **4-insertions-3-3** | ✓ | 1 | 11 | 10 |
| | | | | |
| **Myciel** | | | | |
| **myciel3-3** | ✓ | 1 | 2 | 1 |
| **myciel3-4** | ✓ | 0 | 2 | 2 |
| **myciel4-4** | ✓ | 1 | 5 | 4 |
| **myciel4-5** | ✓ | 0 | 5 | 5 |
| **myciel5-4** | ✓ | 4 | 10 | 6 |
| **myciel5-5** | ✓ | 1 | 9 | 8 |
| **myciel5-6** | ✓ | 0 | 12 | 12 |
| **myciel6-4** | ✓ | 16 | 44 | 28 |
| **myciel6-5** | ✓ | 4 | 32 | 28 |
| **myciel6-6** | ✓ | 1 | 33 | 32 |
| **myciel6-7** | ✓ | 0 | 30 | 30 |
| **myciel7-4** | ✓ | 56 | 101 | 45 |
| **myciel7-5** | ✓ | 20 | 49 | 29 |
| **myciel7-6** | ✓ | 5 | 51 | 46 |
| **myciel7-7** | ✓ | 1 | 56 | 55 |
| **myciel7-8** | ✓ | 0 | 56 | 56 |
| | | | | |
| **Mug** | | | | |

| | | | | |
|---|---|---|---|---|
| **mug88-1-3** | ✓ | 1 | 21 | 20 |
| **mug88-1-4** | ✓ | 0 | 15 | 15 |
| **mug88-25-3** | ✓ | 1 | 25 | 24 |
| **mug88-25-4** | ✓ | 0 | 10 | 10 |
| **mug100-1-3** | ✓ | 1 | 28 | 27 |
| **mug100-1-4** | ✓ | 0 | 19 | 19 |
| **mug100-25-3** | ✓ | 1 | 29 | 28 |
| **mug100-25-4** | ✓ | 0 | 14 | 14 |

Από τα αποτελέσματα της τρίτης στήλης παρατηρείται ότι τα αρχεία games120-9, miles250-8, queen10-10-12, queen11-11-12, myciel3-4, myciel4-5, myciel5-6, myciel6-7, myciel7-8, mug88-1-4, mug88-25-4, mug100-4 και mug100-25-4 έχουν τελικό κόστος 0, οπότε συμπεραίνεται ότι βρέθηκε ανάθεση τιμών των μεταβλητών που αποτελεί λύση του προβλήματος. Από αυτά τα αρχεία, τα αρχεία myciel3-4 (Διαφορά = 2) και myciel4-5 (Διαφορά = 5) πλησιάζουν περισσότερο σε ολοκληρωμένη ανάθεση τιμών με υψηλές αρχικές και τελικές θερμοκρασίες. Οι περιορισμοί είναι της μορφής X≠Y, οπότε ο AC3 δεν διαγράφει τιμές από τα πεδία ορισμών.

Τα προβλήματα έτρεξαν σε tdm-gcc 9.2.0 compiler.

## 6.2 Αποτελέσματα RLFAP προβλημάτων

Παρακάτω παρουσιάζονται τα αποτελέσματα των RLFAP(Radio Links Frequency Assignment Problem) προβλημάτων στα οποία χρησιμοποιήθηκαν οι υβριδικοί αλγόριθμοι για να ελεγχθούν αν είναι arc consistent και για την ανάθεση τιμών. Το πρόβλημα RLFAP συνίσταται στην εκχώρηση συχνοτήτων σε ένα σύνολο ραδιοζεύξεων που ορίζονται μεταξύ ζευγαριών τοποθεσιών, προκειμένου να αποφευχθούν παρεμβολές. Κάθε ραδιοσύνδεσμος αντιπροσωπεύεται από μια μεταβλητή της οποίας ο τομέας είναι το σύνολο όλων των συχνοτήτων που είναι διαθέσιμες για αυτόν τον σύνδεσμο. Οι περιορισμοί είναι της μορφής |X-Y| > k ή |X-Y| = k, όπου k ένας σταθερός αριθμός που εξαρτάται από τη θέση των δύο συνδέσμων και από το φυσικό περιβάλλον.

### 6.2.1 Αποτελέσματα Simulated Annealing

Τα αποτελέσματα της δεύτερης στήλης είναι με χαμηλή αρχική και τελική θερμοκρασία ($T_{αρχική}$ = 0.2, $T_{min}$ = 0.0001, a=0.9998). Τα αποτελέσματα της τρίτης στήλης είναι με αρχική θερμοκρασία $T_{αρχική}$ = 1 και τελική θερμοκρασία $T_{min}$ = 0.1 (a=0.99).

| | Κόστος Small | Κόστος Big | Διαφορά |
|---|---|---|---|
| **2-f24** | 0 | 112 | 112 |
| **2-f25** | 4 | 118 | 114 |
| **6-w2** | 21 | 142 | 121 |
| **7-w1-f4** | 6 | 236 | 230 |
| **7-w1-f5** | 8 | 233 | 225 |
| **11** | 1 | 399 | 398 |

Από τα αποτελέσματα της δεύτερης στήλης, παρατηρούμε ότι το παράδειγμα 2-f24 έχει τελικό κόστος 0, οπότε συμπεραίνουμε ότι σε αυτό το πρόβλημα βρέθηκε λύση. Στην τέταρτη στήλη παρατηρούμε ότι τα

παραδείγματα που είχαν την μικρότερη απόκλιση είναι τα 2-f24 (Διαφορά = 112) και 2-f25 (Διαφορά = 114).

Τα προβλήματα έτρεξαν σε tdm-gcc 9.2.0 compiler.

### 6.2.2 Πρώτος Υβριδικός Αλγόριθμος

Τα αποτελέσματα της τρίτης στήλης είναι με χαμηλή αρχική και τελική θερμοκρασία ($T_{αρχική}$ = 0.2, $T_{min}$ = 0.0001, a=0.9998). Τα αποτελέσματα της τέταρτης στήλης είναι με αρχική θερμοκρασία $T_{αρχική}$ = 1 και τελική θερμοκρασία $T_{min}$ = 0.1 (a=0.99).

| | Arc Consistent | Κόστος Small | Κόστος Big | Διαφορά |
|---|---|---|---|---|
| **2-f24** | ✓ | 0 | 108 | 108 |
| **2-f25** | ✓ | 2 | 112 | 110 |
| **6-w2** | ✓ | 16 | 147 | 131 |
| **7-w1-f4** | ✓ | 5 | 224 | 219 |
| **7-w1-f5** | ✓ | 7 | 248 | 241 |
| **11** | ✓ | 0 | 380 | 380 |

Από τα αποτελέσματα της τρίτης στήλης, παρατηρούμε ότι τα παραδείγματα 2-f24 και 11 έχουν τελικό κόστος 0, οπότε συμπεραίνουμε ότι σε αυτά τα προβλήματα βρέθηκε λύση. Στην πέμπτη στήλη παρατηρούμε ότι τα παραδείγματα που είχαν την μικρότερη απόκλιση είναι τα 2-f24 (Διαφορά = 108) και 2-f25 (Διαφορά = 110).

Τα προβλήματα έτρεξαν σε tdm-gcc 9.2.0 compiler.

### 6.2.3 Δεύτερος Υβριδικός Αλγόριθμος

Τα αποτελέσματα της τρίτης στήλης είναι με χαμηλή αρχική και τελική θερμοκρασία ($T_{αρχική}$ = 0.2, $T_{min}$ = 0.0001, a=0.9998). Τα αποτελέσματα της τέταρτης στήλης είναι με αρχική θερμοκρασία $T_{αρχική}$ = 1 και τελική θερμοκρασία $T_{min}$ = 0.1 (a=0.99).

| | Arc Consistent | Κόστος Small | Κόστος Big | Διαφορά |
|---|---|---|---|---|
| **2-f24** | ✓ | 0 | 74 | 74 |
| **2-f25** | ✓ | 2 | 82 | 80 |
| **6-w2** | ✓ | 16 | 121 | 105 |
| **7-w1-f4** | ✓ | 5 | 191 | 186 |
| **7-w1-f5** | ✓ | 7 | 204 | 197 |
| **11** | ✓ | 0 | 336 | 336 |

Από τα αποτελέσματα της τρίτης στήλης, παρατηρούμε ότι τα παραδείγματα 2-f24 και 11 έχουν τελικό κόστος 0, οπότε συμπεραίνουμε ότι σε αυτά τα προβλήματα βρέθηκε λύση. Στην πέμπτη στήλη παρατηρούμε ότι τα παραδείγματα που είχαν την μικρότερη απόκλιση είναι τα 2-f24 (Διαφορά = 74) και 2-f25 (Διαφορά = 154).

Τα προβλήματα έτρεξαν σε tdm-gcc 9.2.0 compiler.

### 6.2.4 Τρίτος Υβριδικός Αλγόριθμος

Τα αποτελέσματα της τρίτης στήλης είναι με χαμηλή αρχική και τελική θερμοκρασία ($T_{αρχική}$ = 0.2, $T_{min}$ = 0.0001, a=0.9998). Τα αποτελέσματα της τέταρτης στήλης είναι με αρχική θερμοκρασία $T_{αρχική}$ = 1 και τελική θερμοκρασία $T_{min}$ = 0.1 (a=0.99).

| | Arc Consistent | Κόστος Small | Κόστος Big | Διαφορά |
|---|---|---|---|---|
| **2-f24** | ✓ | 0 | 53 | 53 |
| **2-f25** | ✓ | 2 | 64 | 62 |
| **6-w2** | ✓ | 16 | 104 | 88 |
| **7-w1-f4** | ✓ | 5 | 140 | 135 |
| **7-w1-f5** | ✓ | 7 | 131 | 124 |
| **11** | ✓ | 0 | 276 | 276 |

Από τα αποτελέσματα της τρίτης στήλης, παρατηρούμε ότι τα παραδείγματα 2-f24 και 11 έχουν τελικό κόστος 0, οπότε συμπεραίνουμε ότι σε αυτά τα προβλήματα βρέθηκε λύση. Στην πέμπτη στήλη παρατηρούμε ότι τα παραδείγματα που είχαν την μικρότερη απόκλιση είναι τα 2-f24 (Διαφορά = 53) και 2-f25 (Διαφορά = 62).

Τα προβλήματα έτρεξαν σε tdm-gcc 9.2.0 compiler.

## 6.3 Συμπεράσματα Αποτελεσμάτων

Αρχικά χρησιμοποιείται ο απλός Simulated Annealing για την ανάθεση τιμών. Σκοπός των συγκεκριμένων αποτελεσμάτων είναι η σύγκριση τους με τα αποτελέσματα των υβριδικών αλγορίθμων. Υπολογίζεται ότι ο μέσος όρος διαφοράς για τον Simulated Annealing είναι $MO_{SA}$ = 200.

Ο πρώτος υβριδικός αλγόριθμος χρησιμοποιεί πρώτα την συνέπεια τόξου για να μειώσει τον χώρο αναζήτησης και στην συνέχεια χρησιμοποιείται ο Simulated Annealing για την ανάθεση τιμών στις μεταβλητές. Από τα

αποτελέσματα υπολογίζεται ότι ο μέσος όρος διαφοράς για τον πρώτο αλγόριθμο είναι $MO_1 = 198,1$.

Ο δεύτερος υβριδικός αλγόριθμος χρησιμοποιεί τον Simulated Annealing για την ανάθεση τιμών στις μεταβλητές και εφαρμόζεται μια περιορισμένη μορφή συνέπειας τόξου κατά την εκτέλεση του. Συγκριτικά με τον πρώτο υβριδικό αλγόριθμο, η άμεση επιρροή της συνέπειας τόξου στην επιλογή τιμής συμβάλλει στην "καλύτερη" ανάθεση τιμών. Από τα αποτελέσματα υπολογίζεται ότι ο μέσος όρος διαφοράς για τον δεύτερο αλγόριθμο είναι $MO_2 = 163$.

Ο τρίτος υβριδικός αλγόριθμος, παρόμοια με τον δεύτερο υβριδικό αλγόριθμο, χρησιμοποιεί τον Simulated Annealing για την ανάθεση τιμών στις μεταβλητές και εφαρμόζεται μια περιορισμένη μορφή συνέπειας τόξου κατά την εκτέλεση του. Σε αντίθεση με τον προηγούμενο αλγόριθμο, η συνέπεια τόξου δεν επηρεάζει στην επιλογή τιμής αλλά στην αποδοχή ανάθεσης μιας τιμής στην μεταβλητή. Με αυτό τον τρόπο, ο τρίτος αλγόριθμος είναι πιο ευέλικτος από τον δεύτερο αλγόριθμο. Από τα αποτελέσματα υπολογίζεται ότι ο μέσος όρος διαφοράς για τον τρίτο αλγόριθμο είναι $MO_3 = 123$.

Παρατηρείται ότι σε πολύ χαμηλές θερμοκρασίες δεν υπάρχουν μεγάλες διαφορές μεταξύ των υβριδικών αλγορίθμων, ενώ παράλληλα και οι τρεις αλγόριθμοι έχουν καλύτερα αποτελέσματα από τον απλό Simulated Annealing. Σε υψηλότερες θερμοκρασίες, ο πρώτος υβριδικός αλγόριθμος παρουσιάζει παρόμοια αποτελέσματα με τον απλό αλγόριθμο το οποίο είναι λογικό, εφόσον η συνέπεια τόξου δεν επηρεάζει άμεσα στην επιλογή τιμών των μεταβλητών. Με την άμεση επιρροή της συνέπειας τόξου, ο δεύτερος υβριδικός αλγόριθμος παρουσιάζει καλύτερα αποτελέσματα από τον πρώτο υβριδικό. Η επιρροή της συνέπειας τόξου στην αποδοχή τιμών καθιστά τον τρίτο υβριδικό αλγόριθμο πιο ευέλικτο και φαίνεται από τα αποτελέσματα του συγκριτικά με τον δεύτερο υβριδικό. Συμπεραίνεται ότι ο τρίτος υβριδικός αλγόριθμος φαίνεται να αποδίδει καλύτερα σε διαφορετικούς συνδυασμούς παραμέτρων.

# Κεφάλαιο 7

# Συμπεράσματα και επεκτάσεις

Στην παρούσα διπλωματική εξετάστηκαν και υλοποιήθηκαν υβριδικοί αλγόριθμοι της συνέπειας τόξου και του Simulated Annealing για την επίλυση προβλημάτων ικανοποίησης περιορισμών. Ο προγραμματισμός περιορισμών είναι ένα δυνατό εργαλείο για την επίλυση τους. Η επιρροή της συνέπειας τόξου στην αποδοχή τιμών επιτρέπει στην υλοποίηση περισσότερων υβριδικών αλγορίθμων με διαφορετικό κριτήριο από την καταμέτρηση των συνολικών διαγραφών των τιμών. Σε μεταγενέστερο στάδιο, σκοπός είναι η εξέλιξη της παρούσας εργασίας με την μελέτη και προσθήκη περισσότερων υβριδικών αλγορίθμων. Η τροποποίηση του κώδικα για την δυνατότητα επίλυσης προβλημάτων με περιορισμούς που επιβάλλονται σε τέσσερεις ή περισσότερες μεταβλητές αποτελεί μια πιθανή μελλοντική προσθήκη (τα αρχεία Graph Coloring είχε περιορισμούς με δύο μεταβλητές και τα RFLAP με τρεις).

# Βιβλιογραφία

1. Kumar V., Algorithms for Constraint Satisfaction Problems: A Survey, AI Magazine, 13 (1992)

2. Stuart Russell; Peter Norvig “Τεχνητή Νοημοσύνη, Μια σύγχρονη προσέγγιση” (ISBN 9789602098738)

3. Sally C. Brailsford, Chris N. Potts, Barbara M. Smith “Constraint satisfaction problems: Algorithms and applications”

4. Sathiyaraj Chinnasamy, M. Ramachandran, M. Amudha, Kurinjimalar Ramu “A Review on Hill Climbing Optimization Methodology”

5. Steven Minton, Andy Philips, Mark D. Johnston, Philip Laird “Minimizing Conflicts: A Heuristic Repair Method for Constraint-Satisfaction and Scheduling Problems”

6. Emilia Golemanova “Declarative Implementations of Search Strategies for Solving CSPs in Control Network Programming”

7. EL GRAOUI EL MEHDI, BENELALLAM IMADE, BOUYAKHF EL HOUSSINE, "A novel Hybrid Search for Minimal Perturbation Problems based on Backjumping and Dynamic backtracking methods"

8. Fahiem Bacchus, Adam Grov “On The Forward Checking Algorithm”

9. Narendra Jussien, Romuald Debruyne, Patrice Boizumault “Maintaining Arc-Consistency within Dynamic Backtracking”

10. Krzysztof R. Apt “The essence of constraint propagation”

11. Alan K. Mackworth “Consistency in networks of relations”

12. Meriem Zouita, Sadok Bouamama, Kamel Barkaoui “Improving genetic algorithm using arc consistency technic”

13. Pascal Van Hentenryck, Yves Deville, Choh-Man Teng “A generic arc-consistency algorithm and its specializations”

14. Yves Deville, Pascal Van Hentenryck “An Efficient Arc Consistency Algorithm for a Class of CSP Problems”

15. Christian Bessière “Arc-consistency and arc-consistency again”

16. M.R.C. van Dongen, J.A. Bowen “Improving Arc-Consistency Algorithms with Double-Support Checks”

17. Christian Bessire, Jean-Charles Rgin “Refining the Basic Constraint Propagation Algorithm”

18. https://en.wikipedia.org/wiki/Metaheuristic

19. S. KIRKPATRICK, C. D. GELATT, JR., M. P. VECCHI “Optimization by Simulated Annealing”

20. https://en.wikipedia.org/wiki/Simulated_annealing

21. Mikkel Syse Grøsland & Vetle Volden-Freberg “Bachelor Thesis - Variants of Simulated Annealing For Solving Constraint Satisfaction Problems”