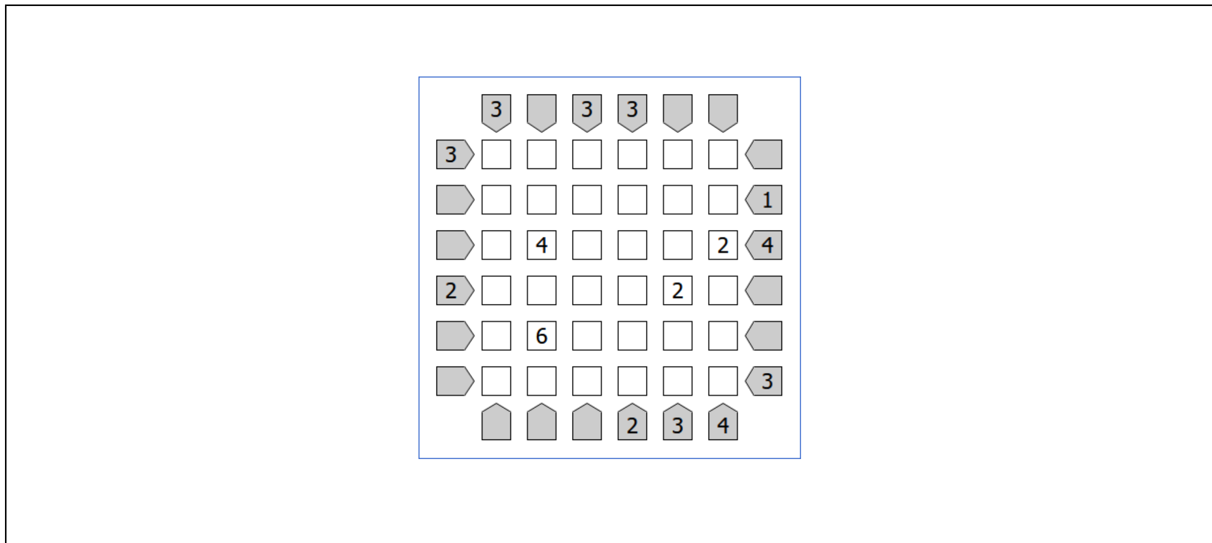
ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«ΕΠΙΛΥΣΗ ΠΑΖΛ ΚΑΙ ΓΡΙΦΩΝ ΜΕ ΤΕΧΝΙΚΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΙΚΑΝΟΠΟΙΗΣΗΣ ΠΕΡΙΟΡΙΣΜΩΝ »



Του φοιτητή
Νικόλαου Μήτσιου
Αρ. Μητρώου: 113750

Επιβλέπων
Ονοματεπώνυμο Ευστάθιος Αντωνίου
Βαθμίδα Καθηγητής

Ημερομηνία 16/01/2023

Τίτλος Π.Ε. Επίλυση παζλ και γρίφων με τεχνικές προγραμματισμού ικανοποίησης περιορισμών
Κωδικός Π.Ε. 21226
Ονοματεπώνυμο φοιτητή Νικόλαος Μήτσιος
Ονοματεπώνυμο εισηγητή Ευστάθιος Αντωνίου
Ημερομηνία ανάληψης Π.Ε. 31/03.2021
Ημερομηνία περάτωσης Π.Ε. 16/01/2023

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του φοιτητή Νικόλαου Μήτσιου που την εκπόνησε. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης, ο συγγραφέας/δημιουργός εκχωρεί στο Διεθνές Πανεπιστήμιο της Ελλάδος άδεια χρήσης του δικαιώματος αναπαραγωγής, δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσης της εργασίας διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος. Η ανοικτή πρόσβαση στο πλήρες κείμενο της εργασίας, δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού, ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, πώληση, εμπορική χρήση, διανομή, έκδοση, μεταφόρτωση (downloading), ανάρτηση (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού.

Η έγκριση της πτυχιακής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

«Αφιέρωση»

Η παρούσα πτυχιακή είναι αφιερωμένη στην ανιψιά μου Αιμιλία.

Πρόλογος

Η παρούσα εργασία εκπονήθηκε το ακαδημαϊκό έτος 2022-2023 στα πλαίσια του προπτυχιακού προγράμματος σπουδών του τμήματος Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων, του Διεθνούς Πανεπιστήμιο της Ελλάδος. Η επίβλεψη της συγγραφής έγινε από τον Καθηγητή Ευστάθιο Αντωνίου. Ο θεματικός άξονας της εργασίας, τα Προβλήματα Ικανοποίησης Περιορισμών, ανήκουν στον τομέα της Τεχνητής Νοημοσύνης καθώς και έχουν καίριο ρόλο στην επίλυση των οργανωτικών (και άλλων) προβλημάτων του ακαδημαϊκού και βιομηχανικού τομέα. Επέλεξα αυτό το θέμα λόγο του μεγάλου ενδιαφέροντος μου για την Τεχνητή Νοημοσύνη σαν τομέα έρευνας. Χάρις στην δυνατότητα της να δίνει γρήγορη και επαρκής λύση σε δύσκολα προβλήματα, όπου η ανάλυση δεν αρκεί, την θεωρώ ένα αναπόσπαστο εργαλείο ενός μοντέρνου μηχανικού. Μέσα από την εκπόνηση αυτής της εργασίας θεωρώ πως θα αποκτήσω σημαντικά εφόδια και γνώσεις που θα βοηθήσουν με την ένταξή στην αγορά εργασίας.

Περίληψη

Τα Προβλήματα Ικανοποίησης Περιορισμών είναι ένας από τους τομείς έρευνας της Τεχνητής νοημοσύνης. Αποτελούν ένα ισχυρό εργαλείο οργάνωσης και λύσης πραγματικών προβλημάτων όπως είναι τα προβλήματα χρονοπρογραμματισμού και δρομολόγησης. Σε αυτή την εργασία παρουσιάζεται μια πλήρης ανασκόπηση του τομέα των προβλημάτων ικανοποίησης περιορισμών: πως μοντελοποιούνται τα προβλήματα με μαθηματικά μοντέλα και πως λύνονται με αλγορίθμους. Δίνεται ιδιαίτερη έμφαση στις τεχνικές, οι οποίες χρησιμοποιούνται κατά την εκτέλεση των προγραμμάτων επίλυσης και πως αυτές βοηθούν στο να επιταχυνθεί η αναζήτηση λύσεων στους συχνά τεράστιους χώρους καταστάσεων. Παρουσιάζονται οι χώροι αναζήτησης "δέντρου" και "πλήρων καταστάσεων" και οι αλγόριθμοι που τους διασχίζουν ψάχνοντας μια λύση. Έπειτα, στο δεύτερο κεφάλαιο, δίνονται παραδείγματα μοντέρνων πραγματικών προβλημάτων στην βιβλιογραφία. Τα προβλήματα αυτά λύνονται με αλγορίθμους ικανοποίησης περιορισμών οι οποίοι παρουσιάζονται στο πρώτο κεφάλαιο. Τέλος γίνεται υλοποίηση ενός μοντέλου του παιχνιδιού "Skyscrapers". Αναλύονται πλήρως οι μεταβλητές και περιορισμοί του μοντέλου και δίνεται αλγόριθμος για την λύση του. Το παιχνίδι λύνεται με ένα πρόγραμμα, υλοποιημένο σε γλώσσα Python με την χρήση των εργαλείων OR-Tools, που σχεδιάστηκαν από την Google για την επίλυση προβλημάτων ικανοποίησης περιορισμών.

«Solving Puzzles and Riddles with Constraint Satisfaction Programming Techniques »

«Nikolaos Mitsios»

Abstract

Constraint Satisfaction Problems is one of the many sectors of research, under the domain of Artificial Intelligence. Constraint Satisfaction Problems are a powerful tool for the organization and solution of real-world problems such as the scheduling problem and the routing problem. In this report we present a brief review of the domain of Constraint Satisfaction Problems: how the problems are modeled using mathematical models and how they are solved using algorithms. Special emphasis is given to the techniques used by the solver programs to accelerate searching in the usually huge search spaces. The "tree" and "complete assignment" search spaces are introduced as well as the algorithms that navigate them in search of a solution. In the second chapter we give examples of how modern real-world problems are solved in the literature. These problems show how the algorithms and techniques of the first chapter are put to use. Finally, we implement the mathematical model of the "Skyscrapers" game. We fully analyze the variables and constraints and construct an algorithm to solve it. A program is developed for solving the game, written in Python and using the OR-Tools library, which was developed by Google for solving constraint satisfaction problems.

Ευχαριστίες

Αρχικά θέλω να εκφράσω τις θερμές μου ευχαριστίες στον καθηγητή μου, κο Ευστάθιο Αντωνίου για την εμπιστοσύνη που μου έδειξε για την ανάθεση της παραπάνω πτυχιακής εργασίας.

Θερμές ευχαριστίες απευθύνω σε όλους τους καθηγητές που είχα όλα τα χρόνια της μέχρι στιγμής ακαδημαϊκής μου ζωής, για τις γνώσεις που μου μετέδωσαν και με έκαναν καλύτερο άνθρωπο.

Τέλος, ένα μεγάλο ευχαριστώ αξίζουν οι γονείς μου Χρήστος και Ρούλα και η αδερφή μου Μαρία, που με στηρίζουν όλα αυτά τα χρόνια, δίνοντάς μου κουράγιο να προχωρώ και να υπερπηδώ κάθε εμπόδιο για να φτάσω στο στόχο μου .

Περιεχόμενα

Πρόλογος ... 6

Περίληψη ... 7

Abstract ... 8

Ευχαριστίες ... 9

Περιεχόμενα ... 10

Κατάλογος Σχημάτων ... 12

Κατάλογος Πινάκων ... 12

Συντομογραφίες ... 13

Κεφάλαιο 1ο: Προβλήματα Ικανοποίησης Περιορισμών ... 15

1.1 Εισαγωγή ... 15

1.2 Μαθηματική Μοντελοποίηση ... 15

1.3 Αναζήτηση σε δέντρο ... 17

1.3.1 Τεχνικές Συνέπειας ... 19

1.3.2 Τεχνικές Διάταξης ... 23

1.3.3 Στρατηγική Οπισθοχώρηση ... 25

1.3.4 Προώθηση Περιορισμών ... 27

1.4 Αναζήτηση Πλήρων Αναθέσεων ... 28

1.4.1 Τοπική Αναζήτηση ... 29

1.5 Προβλήματα Ικανοποίησης - Βελτιστοποίησης Περιορισμών ... 35

1.6 Προβλήματα Ελαστικών Περιορισμών ... 35

1.1 Συμπεράσματα ... 38

Κεφάλαιο 2ο: Παραδείγματα εφαρμογών CSP ... 39

2.1 Εισαγωγή ... 39

2.2 Το Πρόβλημα του Χρονοπρογραμματισμού ... 39

2.3 Σχεδίαση Υλικού ... 44

2.4 Γλώσσες Μοντελοποίησης ... 45

2.5 Παιχνίδια – Παζλ ... 47

2.6 Συμπεράσματα ... 47

Κεφάλαιο 3ο: Μοντελοποίηση του παιχνιδιού “Skyscrapers” ... 49

3.1 Εισαγωγή ... 49

3.2 Κανόνες του “Skyscrapers” ... 49

3.3 Μαθηματική μοντελοποίηση ... 52
3.4 Προγραμματιστικό μοντέλο ... 56
3.5 Υλοποίηση προγραμματιστικού μοντέλου ... 60
3.6 Συμπεράσματα ... 68
Κεφάλαιο 4ο: Συμπεράσματα ή/και προτάσεις βελτίωσης ... 69
ΒΙΒΛΙΟΓΡΑΦΙΑ ... 70
ΠΑΡΆΡΤΗΜΑ Α: Κώδικας skyscrapers_solver.py ... 75
ΠΑΡΆΡΤΗΜΑ Β: Κώδικας skyscrapers_tester.py ... 78

Κατάλογος Σχημάτων

Σχήμα 1.1: Δέντρο καταστάσεων του προβλήματος N-βασιλισσών 17
Σχήμα 1.2: Ένα παράδειγμα λειτουργίας των NC και AC. 18
Σχήμα 1.3: Παράδειγμα γραφήματος μεταβλητών για μεθόδους CT. 20
Σχήμα 1.4: Γράφημα μεταβλητών μετά από δυαδικοποίηση περιορισμών. 20
Σχήμα 1.5: Παράδειγμα συνέπειας κακής διάταξης τιμών. 22
Σχήμα 1.6: Ένα παράδειγμα προβλήματος χρωματισμού. 24
Σχήμα 1.7: Σχηματικό παράδειγμα τεχνικών αναζήτησης σε δέντρο. 26
Σχήμα 1.8 Γραφικό παράδειγμα χώρου τοπικής αναζήτησης. 28
Σχήμα 1.9: Παράδειγμα νευρωνικού GENET. 31
Σχήμα 1.10: Σχεδιάγραμμα των γενετικών αλγορίθμων. 32
Σχήμα 1.11: Παράδειγμα λειτουργίας της P-BB. 34
Σχήμα 2.12: Διαδικασίες συνδυασμού και μετάλλαξης στην εργασία [51]. 39
Σχήμα 2.13: Παράδειγμα αλυσίδας Kempe από την [57]. 40
Σχήμα 2.14: Παραδείγματα από αποτελέσματα εφαρμογών floorplanning [43]. 42
Σχήμα 2.15: Παράδειγμα του παραθύρου του Oz Explorer [66]. 45
Σχήμα 3.16: Ασυμπλήρωτο πλέγμα Skyscrapers 4x4. 46
Σχήμα 3.17: Οι πλευρές “αριστερά” (a), “πάνω” (b), “κάτω” (c) και “δεξιά” (d). 47
Σχήμα 3.18: Παράδειγμα λύσης ενός γρίφου 4x4. 48
Σχήμα 3.19: Ορατότητα από αριστερά και από δεξιά. 48
Σχήμα 3.20: Συμπλήρωση μιας γραμμής. 48
Σχήμα 3.21: Επιλογή μεταβλητής. 55
Σχήμα 3.22: Η γραμμή και η στήλη της επιλεγμένης μεταβλητή. 56
Σχήμα 3.23: Προώθηση περιορισμών. 56
Σχήμα 3.24: Παράδειγμα του λεξικού για N=4. 60
Σχήμα 3.25: Έξοδος του προγράμματος skyscrapers_solver.py. 62
Σχήμα 3.26: Γραφική παράσταση των χρόνων εκτέλεσης. 63

Κατάλογος Πινάκων

Πίνακας 1.1: Οι αλγόριθμοι τοπικής αναζήτησης και οι εφαρμογές τους. 35
Πίνακας 3.2: Επιλογές OR-Tools για στρατηγικές επιλογής μεταβλητής-τιμής [69] 61
Πίνακας 3.3: Χρόνος λύσης χωρίς προώθηση των περιορισμών C^1 (sec). 63
Πίνακας 3.4: Χρόνος λύσης με προώθηση των περιορισμών C^1 (sec). 63
Πίνακας 3.5: Χρόνος δημιουργίας λεξικού μεταθέσεων (sec). 63
Πίνακας 3.6: Χρόνοι λύσης με διάφορες στρατηγικές επιλογής μεταβλητής. 64
Πίνακας 3.7: Χρόνοι λύσης με διάφορες στρατηγικές επιλογής τιμής. 64

Συντομογραφίες

AC	Arc Consistency
BZ	Brelaz Heuristic
B&B	Branch and Bound
CP	Constraint Propagation
CT	Consistency Techniques
CSOP	Constraint Satisfaction Optimization Problems
CSP	Constraint Satisfaction Problems
DFBB	Depth First Branch and Bound
DAC	Directional Arc Consistency
DFS	Depth First Search
DPC	Directional Path Consistency
FF	Fail First
FIFO	First In First Out
GA	Genetic Algorithm
HC	Hill Climbing
HF	Heuristic Function
MAC	Maintaining Arc Consistency
MC	Min-Conflicts
MLP	Multilayer Perceptron
OR	Operational Research
RPC	Restricted Path Consistency
RTL	Register Transfer Level
RW	Random Walk
KC	K-Consistency
PC	Path Consistency
P-ACC	Partial Arc Consistency Check
P-BJ	Partial Backjumping
P-FC	Partial Forward Checking
P-BB	Partial Branch and Bound
SA	Simulated Annealing

SCSP	Soft Constraint Satisfaction Problems
TS	Tabu Search
NC	Node Consistency

Κεφάλαιο 1ο: Προβλήματα Ικανοποίησης Περιορισμών

1.1 Εισαγωγή

Τα Προβλήματα Ικανοποίησης Περιορισμών (CSP - Constraint Satisfaction Problems) είναι ένα υποσύνολο των προβλημάτων του τομέα της Τεχνητής Νοημοσύνης. Οι μέθοδοι επίλυσης CSP είναι ένα πολύ δυνατό εργαλείο εύρεσης λύσεων προβλημάτων του πραγματικού κόσμου και η χρήση των τεχνικών για την επίλυση τους είναι ευρέως διαδεδομένη [3]. Προβλήματα όπως η δρομολόγηση τραίνων, ο προγραμματισμός αποστολών εμπορεύματος, η ανάθεση πόρων σε μία γραμμή παραγωγής είναι καθημερινά προβλήματα που οι άνθρωποι καλούνται να λύσουν. Η μοντελοποίηση τέτοιων προβλημάτων με περιορισμούς βοηθάει στην οργανωμένη, γρήγορη και αυτοματοποιημένη λύση τους από υπολογιστές. Καθώς τα προβλήματα αυτά είναι κατά κύριο λόγο NP-Complete [1-3], η λύση τους σε λογικό χρόνο απαιτεί όχι μόνο την χρήση αλγορίθμων αλλά και την επιστράτευση έξυπνων τεχνικών για την αύξηση απόδοσης τους.

Χωρίζουμε την μοντελοποίηση των CSP σε τρία στάδια: την Μαθηματική Μοντελοποίηση, τον Προγραμματισμό Ικανοποίησης Περιορισμών και την ανάπτυξη προγράμματος Η/Υ. Το τελικό πρόγραμμα αποτελεί εργαλείο με την χρήση του οποίου θα λυθεί το πραγματικό πρόβλημα.

1.2 Μαθηματική Μοντελοποίηση

Μέσω της μαθηματικής μοντελοποίησης παράγεται ένα μαθηματικό μοντέλο το οποίο είναι μία προσέγγιση του προβλήματος. Το μαθηματικό μοντέλο ορίζεται από δύο σύνολα: το σύνολο των μεταβλητών και το σύνολο των περιορισμών. Η μοντελοποίηση ενός προβλήματος σε ένα πρόβλημα ικανοποίησης περιορισμών περιλαμβάνει την αντιστοίχιση των "εννοιών-κομματιών" του προβλήματος σε μεταβλητές και περιορισμούς. Το πρόβλημα πρέπει μέσα από την μοντελοποίηση να "μεταφραστεί" από γενικές λεκτικές έννοιες σε μαθηματικές μεταβλητές και περιορισμούς.

Για παράδειγμα σε ένα πρόβλημα όπου πρέπει να ανατεθούν αίθουσες διδασκαλίας σε διαλέξεις [1], οι "αίθουσες" θα μοντελοποιηθούν σαν μεταβλητές. Προφανώς το πρόβλημα θα έχει και περιορισμούς όπως "δεν μπορούν δύο διαλέξεις να πάρουν μέρος στην ίδια αίθουσα" ή "μία διάλεξη δεν μπορεί να τελείται σε δύο διαφορετικές αίθουσες" οι οποίοι πρέπει να μετασχηματιστούν από φυσική γλώσσα σε μαθηματικούς περιορισμούς.

Το σύνολο των μεταβλητών περιλαμβάνει τις μεταβλητές του προβλήματος [26], μία ανάθεση των μεταβλητών μπορεί να αποτελεί μία λύση του προβλήματος. Αυτές ή μία από αυτές τις αναθέσεις καλούνται να βρουν οι επιστήμονες που χρησιμοποιούν τα CSP. Η κάθε μεταβλητή μπορεί να πάρει μία, και μόνο, τιμή από το πεδίο ορισμού της. Τα πεδία ορισμού συνήθως είναι διακριτά, και πεπερασμένα. Αν τα πεδία ορισμού είναι συνεχή ή άπειρα τα προβλήματα εντάσσονται συνήθως στην κατηγορία των προβλημάτων βελτιστοποίησης Περιορισμών.

Το σύνολο των περιορισμών περιλαμβάνει τους περιορισμούς που ορίζουν ποιες αναθέσεις τιμών είναι σωστές για τις μεταβλητές [26]. Ο κάθε περιορισμός ορίζεται πάνω σε ένα υποσύνολο των μεταβλητών και προσδιορίζει ποιες από όλες τις πιθανές αναθέσεις του συνόλου αυτού είναι αποδεκτές. Αν ο περιορισμός δρα πάνω σε ένα υποσύνολο από Ν μεταβλητές τότε λέγεται Ν-αδικός περιορισμός (μοναδικός για μία μεταβλητή, δυαδικός για δύο κ.ο.κ). Μία ανάθεση τιμών στις μεταβλητές που ικανοποιεί όλους τους περιορισμούς είναι αποδεκτή λύση για το πρόβλημα ικανοποίησης περιορισμών.

Υπάρχουν δύο τρόποι να μοντελοποιηθεί ένας περιορισμός. Ο ένας είναι να οριστούν όλες οι μεταβλητές στις οποίες αναφέρεται ο περιορισμός σε μία οργανωμένη σειρά. Έπειτα πρέπει να δοθούν πλειάδες με αναθέσεις για κάθε μία από αυτές τις μεταβλητές, όπου η θέση της ανάθεσης στην πλειάδα αντιστοιχεί στην θέση της μεταβλητής στην σειρά. Για παράδειγμα οι μεταβλητές μπορεί να είναι {V1, V2, V3,} και το σύνολο με τις αναθέσεις τους {(1,2,3), (1,3,2), (2,3,4), (2,4,3)}. Προφανώς πρέπει οι αναθέσεις να ανήκουν στο πεδίο ορισμού της αντιστοιχία μεταβλητής. Για να είναι έγκυρος ο περιορισμός πρέπει το σύνολο με τις αναθέσεις να περιέχει όλες τις πιθανές αναθέσεις που είναι αποδεκτές από αυτόν. Αυτός είναι ένας περιορισμός επιτρεπτών αναθέσεων [26].

Ο Δεύτερος τρόπος μοντελοποίησης του περιορισμού είναι μέσω μιας μαθηματικής εξίσωσης που επιβάλλει την σωστή ανάθεση τιμών στις μεταβλητές. Για παράδειγμα ο περιορισμός που δόθηκε παραπάνω θα μπορούσε να γραφτεί σαν δύο εξισώσεις V1 < V2 και V1 < V3. Αυτοί είναι δύο περιορισμοί σχέσης. Οι περιορισμοί σχέσης είναι πιο καλοί στο να περιγράφουν την λογική του περιορισμού, όταν όμως η σχέση γίνεται περίπλοκη (π.χ. μη-γραμμική ή συναρτησιακή) η μορφή επιτρεπτών αναθέσεων μπορεί να είναι πιο γρήγορη στην αποτίμηση [26].

Συνήθως απαιτείται να τηρούνται όλοι οι περιορισμοί στην λύση ενός προβλήματος. Αλλά υπάρχουν και προβλήματα στα οποία αυτό δεν είναι αναγκαίο. Τα προβλήματα που μοντελοποιούνται με CSP μπορούν να χωριστούν σε δύο κλάσεις: προβλήματα ισχυρών περιορισμών (hard constraints) και ελαστικών περιορισμών (soft constraints) [3]. Η πρώτη κλάση είναι η συνήθης όπου πρέπει να τηρηθούν όλοι οι περιορισμοί. Στα προβλήματα με ελαστικούς περιορισμούς υπάρχει μία ιεραρχία περιορισμών στην οποία υπάρχουν κάποιοι ισχυροί περιορισμοί και κάποιοι ελαστικοί. Οι ισχυροί πρέπει να ικανοποιηθούν αναγκαστικά αλλά οι ελαστικοί μπορούν να παραλειφθούν. Μεταξύ δύο λύσεων αυτή που τηρεί τους περισσότερους ελαστικούς περιορισμούς είναι μεγαλύτερης ποιότητας. Ανάλογα με το πρόβλημα που μοντελοποιείται μπορεί οι ελαστικοί περιορισμοί να έχουν κάποιο βάρος, βάσει του οποίου υπολογίζεται μία αντικειμενική τιμή που αντιστοιχεί στην αξία της λύσης.

Αφού μοντελοποιηθεί το πρόβλημα, πρέπει να λυθεί. Η λύση του προβλήματος είναι μία ανάθεση τιμών όλων των μεταβλητών που σέβεται τους όλους τους περιορισμούς. Σε κάποια προβλήματα υπάρχουν παραπάνω από μία λύσεις και αρκεί να βρεθεί μία, αυτή λέγεται αποδεκτή. Κάποιες φορές μία αποδεκτή λύση δεν αρκεί και πρέπει να βρεθούν δύο, παραπάνω ή όλες οι λύσεις. Ακόμη σε κάποιες περιπτώσεις χρειάζεται η λύση να είναι βέλτιστη, αυτό σημαίνει πως πρέπει να βρεθεί μία λύση που βελτιστοποιεί μία αντικειμενική συνάρτηση [4]. Για παράδειγμα να βρεθεί λύση σε ένα παζλ με τις λιγότερες κινήσεις. Πόσες λύσεις πρέπει να βρεθούν και αν υπάρχει αντικειμενική

συνάρτηση για βελτιστοποίηση είναι αποφάσεις που παίρνει ο προγραμματιστής όταν υλοποιεί τον αλγόριθμο αναζήτησης. Αυτές σχετίζονται με το μοντέλο που έχει σχεδιάσει και με το πραγματικό πρόβλημα που θέλει να λύσει.

1.3 Αναζήτηση σε δέντρο

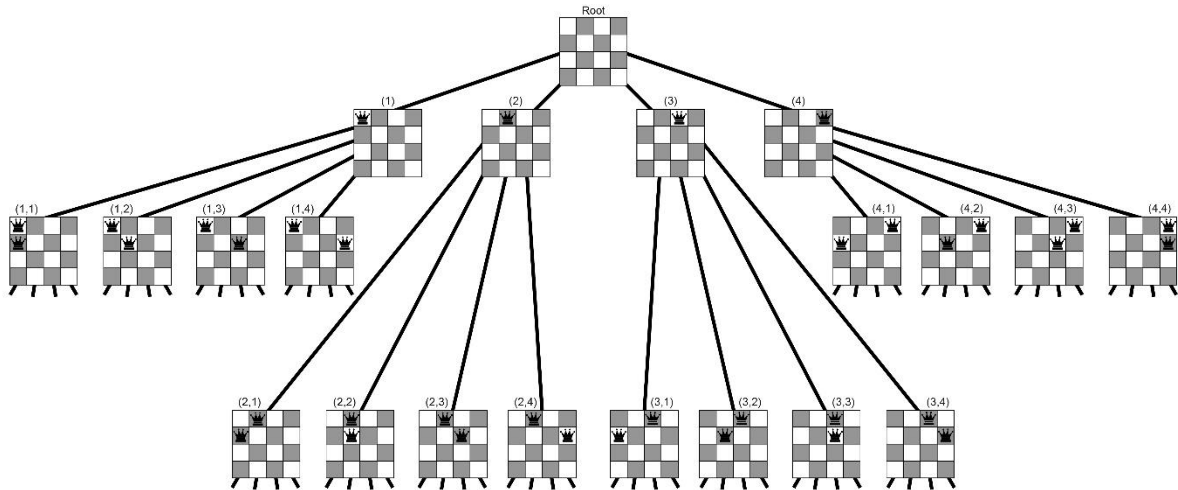
Για να είναι οργανωμένη η εύρεση της λύσης ο αλγόριθμος δουλεύει με καταστάσεις του μοντέλου. Μία κατάσταση ορίζεται από τις μεταβλητές που έχουν συμπληρωθεί με μία τιμή και την τιμή που έχουν πάρει. Η αρχική κατάσταση είναι η κατάσταση στην οποία δεν έχει συμπληρωθεί κάποια μεταβλητή. Δύο καταστάσεις που έχουν τις ίδιες μεταβλητές συμπληρωμένες αλλά με διαφορετικές τιμές είναι διαφορετικές. Γειτονικές είναι δύο καταστάσεις στις οποίες όλες οι τιμές των μεταβλητών είναι ίδιες εκτός από μία. Βήμα είναι η μετάβαση από μία κατάσταση σε μία άλλη (γειτονική) με συμπλήρωση μιας ασυμπλήρωτης μεταβλητής ή διαγραφή μίας συμπληρωμένης.

Με αυτή την οργάνωση οι καταστάσεις του μοντέλου μπορούν να οργανωθούν σε ένα γράφημα όπου κόμβοι είναι οι καταστάσεις και ακμές τα βήματα που συνδέουν τις γειτονικές καταστάσεις. Πιο συγκεκριμένα το γράφημα μπορεί να πάρει την μορφή δέντρου [32], οπού ρίζα είναι η αρχική (κενή) κατάσταση και παιδιά/γονέας κάθε κόμβου αποτελούν οι γειτονικές καταστάσεις του. Παιδιά ενός κόμβου είναι οι καταστάσεις στις οποίες ο αλγόριθμος συμπληρώνει μία ασυμπλήρωτη μεταβλητή, ή αλλιώς κάνει βήμα προς τα εμπρός. Γονέας ενός κόμβου είναι η κατάσταση στην οποία μετέβη διαγράφοντας την τιμή κάποιας μεταβλητής, ή αλλιώς βήμα προς τα πίσω. Οι κόμβοι-φύλλα του δέντρου είναι οι τελικές καταστάσεις του μοντέλου. Στις τελικές καταστάσεις όλες οι μεταβλητές έχουν συμπληρωθεί και το μοντέλο δεν γίνεται να κάνει βήμα προς τα εμπρός καθώς δεν υπάρχει κάποια ασυμπλήρωτη μεταβλητή.

Η ανάθεση τιμών της κάθε κατάστασης στο γράφημα είτε θα παραβιάζει κάποιους περιορισμούς ή θα τηρεί όλους τους περιορισμούς, οι οποίοι μπορούν να εφαρμοστούν στις συμπληρωμένες μεταβλητές. Σε έγκυρα προβλήματα η αρχική κατάσταση δεν παραβιάζει κανέναν περιορισμό. Αν κάποια ενδιάμεση ή τελική κατάσταση παραβιάζεται ένας τουλάχιστον περιορισμός τότε αυτή θεωρείται κατάσταση-αδιέξοδο. Οι τελικές καταστάσεις οι οποίες τηρούν όλους τους περιορισμούς είναι οι αποδεκτές λύσεις του προβλήματος και αποτελούν τον στόχο της αναζήτησης.

Ο αλγόριθμος κάνει μία αναζήτηση στο δέντρο ξεκινώντας από την αρχική κατάσταση και μέσω βημάτων μεταξύ τον καταστάσεων αποσκοπεί στο να φτάσει σε μία ή παραπάνω τελικές καταστάσεις. Η αναζήτηση αυτή βασίζεται στον αλγόριθμο Αναζήτησης κατά Βάθος (DFS - Depth First Search) [32]. Ο DFS είναι ένας αλγόριθμος ο οποίος κάνει την προσπέλαση ενός ολόκληρου δέντρου. Είναι ιδανικός για την αναζήτηση σε δέντρο καθώς δίνει προτεραιότητα στο να φτάσει σε φύλλα από το να εξετάσει όλους τους κόμβους ίδιου επιπέδου. Η αναζήτηση είναι ένας αλγόριθμος τύπου DFS ο οποίος εξετάζει την κάθε κατάσταση που προσπελάζει για να αποφασίσει αν θα σταματήσει την αναζήτηση ή με ποιόν τρόπο θα προχωρήσει. Όπως θα φανεί παρακάτω οι τεχνικές του CSP υλοποιούνται σε αυτό το στάδιο της απόφασης.

Ένα απλό παράδειγμα προβλήματος που λύνεται με μεθόδους CSP είναι το πρόβλημα των Ν-βασιλισσών [25]. Το πρόβλημα ζητάει να τοποθετηθούν Ν βασίλισσες σε μία σκακιέρα διαστάσεων ΝxΝ έτσι ώστε καμία να μην απειλεί κάποια άλλη. Οι μεταβλητές για το μαθηματικό μοντέλο αυτού του προβλήματος είναι Ν μεταβλητές γραμμής. Η μεταβλητές αντιπροσωπεύουν μία γραμμή της σκακιέρας και η τιμή της κάθε μίας δείχνει σε ποια στήλη της σκακιέρας θα τοποθετηθεί η βασίλισσα (θεωρώντας ότι δεν επιτρέπεται να τοποθετηθούν δύο βασίλισσες σε μία γραμμή). Το πεδίο ορισμού όλων των μεταβλητών είναι {1, 2, … , Ν}. Ένα μικρό κομμάτι του δέντρου καταστάσεων αυτού του μοντέλου, για Ν=4, φαίνεται στο σχήμα 1.1.



Σχήμα 11.1: Δέντρο καταστάσεων του προβλήματος Ν-βασιλισσών.

Το δέντρο του σχήματος περιέχει όλες τις πιθανές αναθέσεις των μεταβλητών για το μοντέλο που δόθηκε σαν παράδειγμα. Αυτό το δέντρο θα πρέπει να εξερευνήσει ο αλγόριθμος σε αναζήτηση της λύσης.

Λόγο της πολυπλοκότητας των πραγματικών προβλημάτων που μοντελοποιούνται με CSP το πλήθος των μεταβλητών είναι συνήθως μεγάλο και τα πεδία ορισμού τους εκτενή. Αυτό δημιουργεί μεγάλο αριθμό πιθανών καταστάσεων του μοντέλου αφού σε κάθε κατάσταση πρέπει να επιλεγούν και να εξεταστούν πολλές επόμενες μεταβλητές και για κάθε μία από αυτές όλες οι πιθανές τιμές τους. Έτσι κάθε κόμβος του δέντρου καταλήγει να έχει πολλά παιδία αυξάνοντας το μέγεθος του χώρου αναζήτησης εκθετικά. Αυτό σημαίνει ότι, εκτός από μικρά προβλήματα, είναι σχεδόν αδύνατον να εξεταστούν όλες οι πιθανές καταστάσεις ενός μοντέλου, λόγο του χρόνου που θα έπαιρνε ένα τέτοιο εγχείρημα. Για γίνει η αναζήτηση αποδοτικά έχουν αναπτυχθεί διάφορες τεχνικές οι οποίες μειώνουν τον χώρο αναζήτησης.

Ο λόγος που ο χώρος αναζήτησης είναι μεγάλος είναι διότι πρέπει να ελεγχθεί κάθε πιθανή κατάσταση του μοντέλου [27]. Αλλά αρκετές από αυτές μπορούν να αποφευχθούν. Ανάλογα με το μοντέλο, ειδικά στα σύνθετα μοντέλα για πραγματικά προβλήματα, θα υπάρχουν σχεδόν σίγουρα αδιέξοδα στο δέντρο. Αδιέξοδο είναι μια κατάσταση που παραβιάζει έναν τουλάχιστον (ισχυρό) περιορισμό. Για παράδειγμα στο δέντρο του σχήματος η κατάσταση (1,1) είναι αδιέξοδο: η βασίλισσα της δεύτερης γραμμής δεν θα μπορούσε να τοποθετηθεί σε αυτή την στήλη καθώς απειλεί την

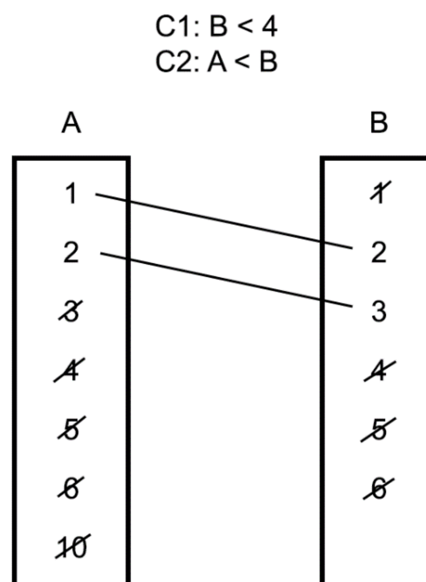
βασίλισσα της πρώτης γραμμής. Υπάρχουν και άλλα αδιέξοδα στο σχήμα 1.1, όπως είναι λογικό δεν θα έχει νόημα να συνεχίσει ο αλγόριθμος να εξετάζει καταστάσεις απογόνους ενός αδιεξόδου καθώς είναι βέβαιο ότι κάθε φύλλο που προήλθε από αυτό θα παραβιάζει επίσης τον περιορισμό.

Είναι σημαντικό ένας αλγόριθμος να μπορεί να αποφεύγει τα αδιέξοδα όχι μόνο όταν τα συναντά αλλά και να προσπαθεί να τα αποφεύγει πριν πέσει πάνω τους επιλέγοντας ένα καλύτερο μονοπάτι. Γι' αυτό αναπτύχθηκαν και χρησιμοποιούνται οι τεχνικές που θα παρουσιαστούν παρακάτω.

1.3.1 Τεχνικές Συνέπειας

Ένα πρώτο βήμα που μπορεί να κάνει κάποιος, πριν τρέξει τον αλγόριθμο αναζήτησης είναι να αφαιρέσει τις ασυνεπείς τιμές από τα πεδία ορισμού των μεταβλητών του μοντέλου. Αυτό γίνεται με τις Τεχνικές Συνέπειας (CT - Consistency Techniques) και η χρήση τους εκτελείται προκαταρκτικά σαν ένα συμπληρωματικό βήμα στην αναζήτηση. Σκοπός τους είναι να αφαιρέσουν καταστάσεις, από τον χώρο αναζήτησης, οι οποίες είτε είναι αδιέξοδα είτε θα οδηγήσουν σε αδιέξοδα. Η αφαίρεση μιας τιμής από το πεδίο ορισμού μιας μεταβλητής ισοδυναμεί με αφαίρεση, από το δέντρο, όλων των καταστάσεων που έχουν αναθέσει αυτή την τιμή στην μεταβλητή. Δυστυχώς σπανίως αρκούν οι CT για την επίλυση του προβλήματος, καθώς δεν αφαίρει εντελώς τα αδιέξοδα από το δέντρο, άρα δεν είναι πλήρης λύση. Παρόλα αυτά οι CT, σε ορισμένες περιπτώσεις μοντέλου, κλαδεύουν το δέντρο και διευκολύνουν τον αλγόριθμο αναζήτησης να βρει την λύση.

Η πιο απλή CT είναι η Συνέπεια Κόμβου (NC - Node Consistency). Αν υπάρχουν μοναδικοί περιορισμοί στο μοντέλο η NC ελέγχει ότι όλες οι τιμές στο πεδίο ορισμού των μεταβλητών που συμμετέχουν στους περιορισμούς αυτούς είναι συνεπείς. Για παράδειγμα αν το μοντέλο έχει μια μεταβλητή X με πεδίο ορισμού {1,2,3,4,5,6,10} και υπάρχει ο περιορισμός "X<4" η NC θα μετατρέψει το πεδίο ορισμού της X σε {1,2,3}.

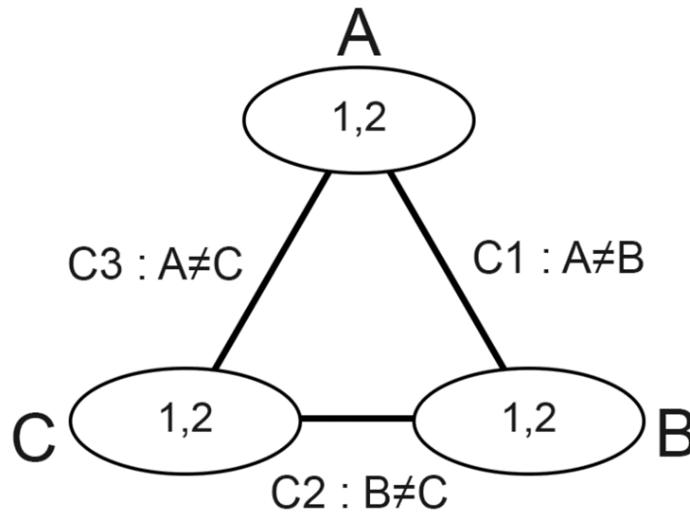


Σχήμα 11.2: Ένα παράδειγμα λειτουργίας των NC και AC.

Φαίνονται τα αρχικά πεδία ορισμού των μεταβλητών A και B. H B έχει τον μοναδικό περιορισμό C1 άρα η NC αφαιρεί τις τιμές (4,5,6). Έπειτα για τον περιορισμό C1 η AC αφαιρεί όλες τις τιμές που δεν υποστηρίζονται. Για κάθε τιμή ο AC αρκεί να βρει μια τιμή στο άλλο πεδίο ορισμού που την υποστηρίζει και να τις ενώσει με μια ακμή. Κάθε τιμή που δεν έχει ακμή δεν υποστηρίζεται και διαγράφεται.

Πιο συνήθης και πιο χρήσιμη τεχνική από την NC είναι η Συνέπεια Τόξου (AC - Arc Consistency) [24]. Η AC αντί για μοναδικούς περιορισμούς δρα πάνω σε δυαδικούς περιορισμούς. Η AC εξετάζει τα ζευγάρια μεταβλητών που συμμετέχουν στους δυαδικούς περιορισμούς και ελέγχει ότι κάθε τιμή στο πεδίο ορισμού τους υποστηρίζεται. Για να υποστηρίζεται μια τιμή στο πεδίο ορισμού μιας μεταβλητής πρέπει να υπάρχει μια τουλάχιστον τιμή στο πεδίο ορισμού της άλλης μεταβλητής έτσι ώστε αν οι δύο τιμές αυτές ανατεθούν στις μεταβλητές να ικανοποιείται ο περιορισμός. Σαν παράδειγμα, δίνεται ένα μοντέλο έχει τις μεταβλητές A και B με πεδία ορισμού {1,2,3,10} και {2,3,4} αντίστοιχα και περιορισμό “A<B”. Τότε η AC θα αφαιρέσει την τιμή “10” από το πεδίο ορισμού της A καθώς δεν υπάρχει καμία τιμή b στο πεδίο ορισμού της B για την οποία να ισχύει ότι “10<b”. Οι πιο γνωστοί αλγόριθμοι που εκτελούν AC σε μοντέλα είναι η σειρά των αλγορίθμων “AC-1” μέχρι “AC-7” [3,4] (οι αλγόριθμοι της σειράς ονομάζονται απλά “AC-x” οπού x είναι ο αριθμός έκδοσης τους).

Ο γενικός τρόπος με το οποίο λειτουργούν οι αλγόριθμοι συνέπειας είναι με την μέθοδο της επαναληπτικής εξισορρόπησης ενός γραφήματος μεταβλητών. Το γράφημα αυτό περιλαμβάνει τις μεταβλητές του μοντέλου σαν κόμβους και σαν ακμές μεταξύ τους τους δυαδικούς περιορισμούς στους οποίους συμμετέχει το κάθε ζευγάρι μεταβλητών. Η εξισορρόπηση περιλαμβάνει την αφαίρεση ασυνεπών τιμών και είναι επαναληπτική γιατί ο αλγόριθμος επανεξετάζει τους κόμβους πολλές φορές. Ο AC-3 [11], παραδείγματος χάρη, ελέγχει τις ακμές μία προς μία για να αφαιρέσει κάθε ασυνεπή τιμή. Μόλις αφαιρέσει μία τιμή από το πεδίο ορισμού κάποιας μεταβλητής ελέγχει ότι δεν δημιουργήθηκε κάποια ασυνέπεια και στις άλλες ακμές που συνδέονται με την μεταβλητή. Όταν δεν υπάρχουν πλέον ασυνέπειες στα πεδία ορισμού και δεν υπάρχουν άλλες ακμές για έλεγχο, το γράφημα είναι ισορροπημένο και ο αλγόριθμος επιστρέφει τα νέα πεδία ορισμού για τις μεταβλητές. Αν τυχαίνει τα πεδία αυτά να περιέχουν μόνο μια τιμή τότε δεν χρειάζεται να εξεταστεί κανένας συνδυασμός και το πρόβλημα λύθηκε, αλλά δεν υπάρχει κάποια εγγύηση ότι θα συμβεί αυτό.

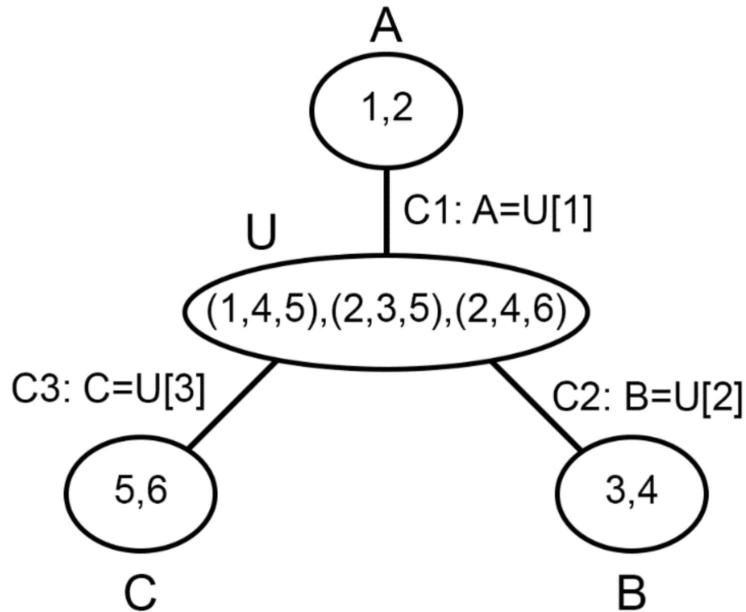


Σχήμα 11.3: Παράδειγμα γραφήματος μεταβλητών για μεθόδους CT.

Στο γράφημα αυτό υπάρχουν τρείς μεταβλητές (A,B,C) και ενώνονται από τους τρείς δυαδικούς περιορισμούς (C1,C2,C3). Η μέθοδος AC θα θεωρούσε όλες τις τιμές συνεπείς καθώς ελέγχει μόνο ζευγάρια μεταβλητών. Η PC όμως θα έβρισκε ότι το πρόβλημα δεν έχει λύση καθώς όλες οι τιμές τις C είναι ασυνεπείς με τις τιμές των A και B.

Μία πιο ισχυρή CT είναι η Συνέπεια Μονοπατιού (PC - Path Consistency) [24]. Στην PC εξετάζεται όχι μόνο η ακμή μεταξύ μεταβλητών αλλά και κάθε μονοπάτι μεταξύ τους [10]. Οι αλγόριθμοι που εκτελούν PC στο γράφημα μεταβλητών ελέγχουν το AC για μία τιμή a μιας μεταβλητής A. Αν βρουν μια τιμή b για την άλλη μεταβλητή B, τότε ελέγχουν ότι σε κάθε μονοπάτι, μεταξύ των A και B, υπάρχουν τιμές για κάθε ενδιάμεσο κόμβο που να ικανοποιούν τους περιορισμούς του μονοπατιού. Αν οι ενδιάμεσες τιμές υπάρχουν τότε αυτό το ζευγάρι a,b είναι path consistent. Για να περιοριστούν οι πόροι που χρειάζεται ο αλγόριθμος ελέγχου PC συνήθως ελέγχονται μονοπάτια μέχρι μεγέθους 2 [3].

Η PC είναι πιο ισχυρή από την AC, καθώς αφαιρεί παραπάνω ασυνεπείς τιμές, αλλά λειτουργεί και αυτή μόνο με δυαδικούς περιορισμούς. Υπάρχει μέθοδος με την οποία μπορεί ένα μοντέλο CSP με N-αδικούς περιορισμούς (για διάφορα N, N>1) να μετατραπεί σε μοντέλο με δυαδικούς περιορισμούς [11]. Συνοπτικά, για κάθε N-αδικό περιορισμό θα δημιουργηθεί μια νέα μεταβλητή U με πεδίο ορισμού τις πλειάδες με τις επιτρεπτές τιμές που τηρούν τον περιορισμό. Οι πλειάδες είναι μήκους N και η κάθε θέση τους αντιστοιχεί σε μια από τις μεταβλητές. Στο γράφημα η νέα μεταβλητή μπορεί να συνδεθεί με τις N μεταβλητές που συμμετέχουν στον περιορισμό με νέους δυαδικούς περιορισμούς. Όταν επιλεγεί μια τιμή για κάποια από τις μεταβλητές του N-αδικού και μια πλειάδα από την U, ο δυαδικός περιορισμός μεταξύ τους απαιτεί η τιμή της μεταβλητής να είναι ίση με την τιμή της θέσης, της πλειάδας, που αντιστοιχεί στην μεταβλητή.



Σχήμα 11.4: Γράφημα μεταβλητών μετά από δυαδικοποίηση περιορισμών.

Το μοντέλο περιέχει τις μεταβλητές (A,B,C) και τον περιορισμό "A+B=C". Ο Τριαδικός περιορισμός μετατρέπεται στην μεταβλητή U και τους τρεις δυαδικούς περιορισμούς (C1, C2, C3) και έτσι μπορεί να φτιαχτεί το γράφημα μεταβλητών.

Ακολουθώντας την ιδέα της δυαδικοποίησης των περιορισμών μπορεί να οριστεί η K-Συνέπεια (KC - K-Consistency) [9]. Η KC μπορεί να επιβεβαιώσει την συνέπεια των τιμών K μεταβλητών για κάθε K-αδικό περιορισμό μεταξύ τους και να επιστρέψει τα K-Συνεπή πεδία ορισμού τους. Αν για ένα γράφημα N μεταβλητών ελεγχθεί η J-Συνέπεια για κάθε J μικρότερο ή ίσο με το N τότε τα πεδία ορισμού των μεταβλητών θα είναι Ισχυρά N-Συνεπή (Strong N-Consistent). Γραφήματα N μεταβλητών που είναι Ισχυρά N-Συνεπή έχουν μόνο συνεπείς τιμές, αρά αποτελούν λύση του προβλήματος [3].

Καθώς αυξάνεται η ισχύς των μεθόδων CT στο να αφαιρούν ασυνεπείς τιμές, αυξάνεται και η πολυπλοκότητα και συνεπώς αυξάνεται εκθετικά το κόστος της χρήσης τους. Παρά το μεγάλο κόστος τους δεν είναι πλήρεις, γι' αυτό έχει γίνει προσπάθεια να γίνουν πιο προσιτοί με την δημιουργία αλγορίθμων που μειώνουν τους πόρους που απαιτούνται αλλά κρατούν σταθερή την απόδοση. Τρία παραδείγματα είναι οι Directional Arc Consistency (DAC), Directional Path Consistency (DPC) και Restricted Path Consistency (RPC) [3].

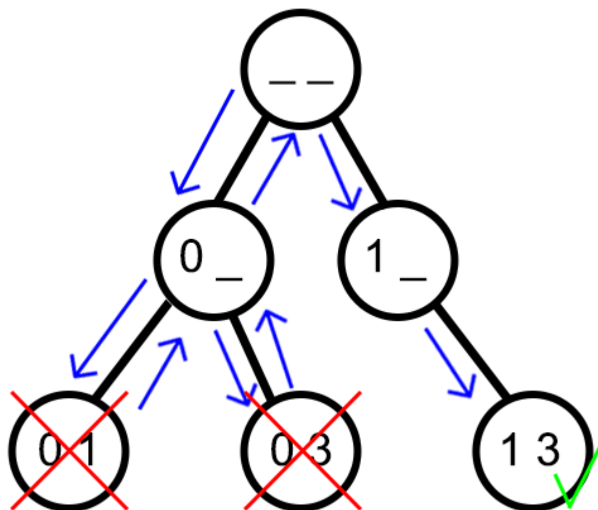
Από τις προαναφερόμενες μεθόδους CT σπάνια χρησιμοποιείται κάτι άλλο από τις NC και AC. Σε μερικά προβλήματα όμως μπορούν να φανούν χρήσιμες, όπως για παράδειγμα η κατηγοριοποίηση ακμών μιας εικόνας [12].

1.3.2 Τεχνικές Διάταξης

Προκειμένου να κάνει το αρχικό βήμα της αναζήτησης, ο αλγόριθμος πρέπει να απαντήσει κάποιες ερωτήσεις. Για να μεταβεί σε κάποια κατάσταση πρέπει αρχικά να επιλέξει μια μεταβλητή και μετά μια τιμή από το πεδίο ορισμού της μεταβλητής αυτής. Έπειτα πρέπει να εξετάσει τους περιορισμούς για να δει αν η δοσμένη τιμή παραβαίνει κάποιον. Η απόφαση της σειράς με την οποία, ο αλγόριθμος, θα επιλέξει τις μεταβλητές, της σειράς με την οποία θα δοκιμάσει τις τιμές από το πεδίο ορισμού τους και τις σειράς με την οποία θα ελέγξει τους περιορισμούς μπορούν να επηρεάσουν δραστικά την απόδοση του. Για να γίνουν οι επιλογές αυτές υιοθετούνται τεχνικές διάταξης.

Η πιο απλή μέθοδος είναι να μην γίνει καθόλου διάταξη και η επιλογή να γίνεται τυχαία. Η τυχαία επιλογή έχει το θετικό ότι δεν σπαταλάει πόρους για να γίνει. Πιο πολύπλοκες μέθοδοι απαιτούν μια Ευρετική Συνάρτηση (HF - Heuristic Function) η οποία θα αναθέσει μια αξία σε κάθε μεταβλητή ανάλογα με το πόσο σημαντικό είναι η ανάθεση της να γίνει σύντομα. Αυτή η HF μπορεί να είναι ακριβή στην χρήση της και κάποια μοντέλα μπορεί να λύνονται πιο γρήγορα χωρίς αυτή. Πέρα από αυτό, η χρήση μιας HF μπορεί να βελτιώσει αρκετά τον χρόνο λύσης [14]. Αυτό γίνεται διότι μια καλή επιλογή μεταβλητής μπορεί να βοηθήσει τον αλγόριθμο να ανακαλύψει αδιέξοδα πιο γρήγορα ή να αποφύγει μονοπάτια που έχουν πολλά αδιέξοδα

Η μέθοδος Fail First (FF) ορίζει ότι, για την επιλογή μεταβλητών και την σειρά με την οποία γίνεται ο έλεγχος περιορισμών, πρέπει να επιλέγεται πρώτα όποιος είναι πιο πιθανό να αποτύχει [13]. Για τον έλεγχο περιορισμών είναι προφανές γιατί ισχύει αυτό: αφού αρκεί ένας περιορισμός να παραβιάζεται για να είναι αδιέξοδο μια κατάσταση δεν χρειάζεται να ελεγχθούν άλλοι περεταίρω. Σε συνδυασμό με το γεγονός ότι έχει επιλεγεί η μεταβλητή που είναι πιο πιθανόν να αποτύχει, όσο πιο γρήγορα αποδειχθεί το αδιέξοδο τόσο το καλύτερο. Η FF σαν ευρετική συνάρτηση για να επιλεγεί η μεταβλητή που είναι πιο πιθανό να αποτύχει, προτείνει την μεταβλητή με το μικρότερο πεδίο ορισμού.



Σχήμα 11.5: Παράδειγμα συνέπειας κακής διάταξης τιμών.

Σε αυτό το δέντρο από τα τρία φύλλα μόνο το ένα είναι αποδεκτή λύση. Οι τιμές διατάσσονται ταξινομημένες και έτσι ο DFS κάνει διπλάσια δουλειά για να φτάσει στην λύση. Αν οι τιμές είχαν πιο καλή διάταξη θα αποφευγόταν οι κόμβοι στα αριστερά.

Μια άλλη τεχνική διάταξης των μεταβλητών είναι η ευρετική Brelaz (BZ) [15]. Παρότι παρουσιάζεται σαν μέθοδος για χρωματισμό γραφήματος, μπορεί να χρησιμοποιηθεί και σε άλλα προβλήματα. Η BZ υιοθετεί την λογική της FF, επιλέγοντας μια μεταβλητή με το μικρότερο τρέχον πεδίο ορισμού, για το υπό χρωματισμό γράφημα, η οποία είναι ο κόμβος με τους περισσότερους ήδη χρωματισμένους γείτονες. Αν τυχαίνει δύο μεταβλητές να είναι ισάξιες στην ευρετική FF τότε η BZ επιλέγει αυτή που συμμετέχει στους περισσότερους περιορισμούς. Η επιλογή αυτή γίνεται βάσει του βαθμού της της κορυφής στο γράφημα μεταβλητών που παρουσιάστηκε στο κεφάλαιο Τεχνικές Συνέπειας. Αν για ένα μοντέλο, η απόκλιση από τον μέσο των βαθμών συμμετοχής σε περιορισμούς είναι μικρή για όλες τις μεταβλητές (π.χ. όλες οι μεταβλητές συμμετέχουν σε K περιορισμούς) τότε η BZ συμπεριφέρεται όπως η FF [16].

Τρεις ακόμα ευρετικές για την διάταξη μεταβλητών είναι η «Ρω» (το Ελληνικό γράμμα), E(N) και «Κάππα» [16]. Και οι τρείς προσπαθούν να επιλέξουν το υπό-πρόβλημα με τις περισσότερες λύσεις, με διαφορετικές ευρετικές συναρτήσεις. Η «Ρω» βασίζεται στην μετρική ρ που είναι η πυκνότητα λύσεων. Επιλέγει την μεταβλητή με την μεγαλύτερη υπολογισμένη πυκνότητα λύσεων στους απογόνους της. Η E(N) προσπαθεί να επιλέξει την μεταβλητή με τους πιο αυστηρούς περιορισμούς, καθώς μια μεταβλητή με πολύ χαλαρούς περιορισμούς θα δημιουργήσει πολλά μονοπάτια στα οποία θα μοιραστούν οι πιθανές λύσεις που την ακολουθούν. Η μέθοδος «Κάππα» χρησιμοποιεί την μετρική της «περιοριστικότητας» (constrainedness) [17], η οποία δείχνει πόσο περιορισμένο είναι ένα πρόβλημα. Προβλήματα που είναι πολύ περιορισμένα ($\kappa>1$) θα είναι πιο δύσκολο να λυθούν, ενώ για πιο χαλαρά προβλήματα ($\kappa<1$) πιο εύκολο. Κάποια μέθοδος εύρεσης του αριθμού πιθανών λύσεων είναι το βασικό στοιχείο αυτών των τριών ευρετικών. Το πλεονέκτημα της «Κάππα» είναι ότι μπορεί να προσαρμοστεί σε πολλά παρόμοια μοντέλα χωρίς να χρειαστεί να υλοποιηθεί από την αρχή

Η μετρική υπόσχεσης (Promise) [19] κάνει διάταξη τιμών μιας μεταβλητής. Για μια επιλεγμένη τιμή α μιας μεταβλητής A μετράει πόσες τιμές μένουν διαθέσιμες στα πεδία ορισμού των άλλων μεταβλητών έτσι ώστε να μην παραβιάζεται κανένας περιορισμός στον οποίο συμμετέχει η A. Υπολογίζει το γινόμενο αυτών των τιμών για κάθε μεταβλητή διαφορετική της A. Αν κάποια μεταβλητή δεν έχει καμία τιμή διαθέσιμη τότε η υπόσχεση επιστρέφει μηδέν, η ευρετική πρέπει να μεγιστοποιηθεί έτσι ώστε να επιλεγεί μια καλή τιμή.

Για την διάταξη τιμών υπάρχουν και οι μετρικές του κόστους (Cost) και κρισιμότητας (Cruciality) [19]. Το κόστος μετράει, σε αντίθεση με την υπόσχεση, πόσες τιμές χάνονται από τα πεδία ορισμού των άλλων μεταβλητών όταν επιλέγει μια τιμή, δηλαδή τις τιμές που θα παραβιάζουν κάποιον περιορισμό αν επιλεγούν στο μέλλον. Αθροίζοντας τις χαμένες τιμές για κάθε άλλη μεταβλητή υπολογίζεται το κόστος. Μία καλή τιμή έχει κόστος κοντά στο μηδέν. Η κρισιμότητα διαιρεί τις

χαμένες τιμές με το πλήθος τιμών στο πεδίο ορισμού της κάθε μεταβλητής. Δηλαδή αυτό που αθροίζεται στην κρισιμότητα είναι το ποσοστό των τιμών που χάθηκαν για κάθε μεταβλητή.

Οι ευρετικές που αναφέρθηκαν μπορούν είτε να εφαρμοστούν μια φορά σαν προ-επεξεργαστικό βήμα στη αρχή της αναζήτησης (στατικά) ή σε κάθε βήμα της αναζήτησης (δυναμικά) μετά από την προώθηση περιορισμών, μια τεχνική που θα συζητηθεί παρακάτω. Αν εφαρμοστούν στατικά δημιουργούν μια διάταξη προτεραιότητας για τις μεταβλητές, τις επιμέρους τιμές κάθε μεταβλητής και των ελέγχων περιορισμών. Στην περίπτωση δυναμικής εφαρμογής η διάταξη προτεραιότητας δημιουργείται μία φορά σε κάθε βήμα, για την επιλογή μιας από τις ασυμπλήρωτες μεταβλητές και για τις τιμές μόνο αυτής της μεταβλητής.

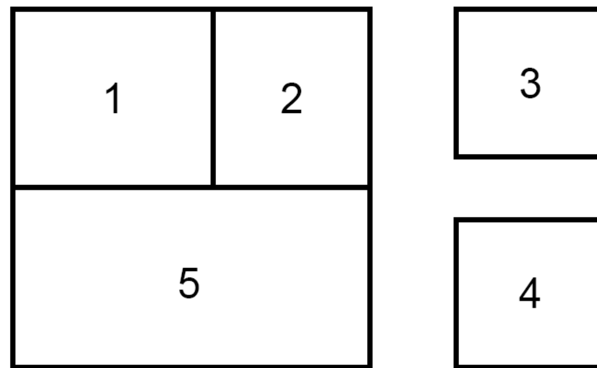
1.3.3 Στρατηγική Οπισθοχώρηση

Όταν ξεκινήσει η αναζήτηση ο αλγόριθμος προσπελάζει το δέντρο ξεκινώντας από την αρχική κατάσταση και μέσω βημάτων μεταξύ τον καταστάσεων έχει στόχο να φτάσει σε μία τελική κατάσταση. Όταν συναντά ένα αδιέξοδο δεν υπάρχει νόημα να συνεχίσει να εξετάζει τις καταστάσεις απογόνους του, οπότε πρέπει να γυρίσει σε κάποια προηγούμενη κατάσταση και να συνεχίσει την αναζήτηση από αυτήν. Σε αντίθεση με αυτή την λογική ο τυπικός DFS, που χρησιμοποιεί η αναζήτηση θα συνέχιζε μέχρι να εξετάσει όλα τα φύλλα-απογόνους του αδιεξόδου.

Η πιο απλή τεχνική οπισθοχώρησης (Backtracking) [24] τροποποιεί τον DFS, έτσι ώστε να σταματά όταν μια κατάσταση είναι αδιέξοδο. Ο DFS συνεχίζει από την αμέσως προηγούμενη κατάσταση, το αδιέξοδο έχει ήδη διαγραφεί από το μέτωπο αναζήτησης άρα δεν θα επανεξεταστεί. Με αυτόν τον τρόπο αποφεύγονται όλες οι καταστάσεις-απόγονοι του αδιεξόδου.

Πολλές φορές στην αναζήτηση προκύπτει ότι μια τιμή που έχει ανατεθεί σε μία μεταβλητή “A”, παρότι δεν παραβαίνει κανέναν περιορισμό, δεν ταιριάζει με καμία από τις πιθανές τιμές κάποιας άλλης μεταβλητής “B”, στην οποία δεν έχει δοθεί τιμή ακόμα. Όταν συμβεί αυτό ο αλγόριθμος θα πρέπει πρώτα να ελέγξει κάθε τιμή της B για να καταλάβει ότι η τιμή που δόθηκε στην A είναι αδιέξοδο. Αυτό το φαινόμενο λέγεται Thrashing [4]. Το Thrashing ρίχνει ιδιαιτέρως την απόδοση όταν η B επιλεγεί για ανάθεση πολύ αργότερα από την A, έχουν δηλαδή επιλεγεί άλλες μεταβλητές ανάμεσα τους.

Για να βελτιωθεί η οπισθοχώρηση φτιάχτηκαν μέθοδοι στις οποίες ο αλγόριθμος αντί για Backtracking στην προηγούμενη κατάσταση κάνει οπισθοχώρηση σε κάποια παλαιότερη. Για να αποφευχθεί το Thrashing η μέθοδος Backjumping ελέγχει ποιες τιμές μεταβλητών, που έχουν ήδη επιλεγεί, συγκρούονται με την τρέχουσα μεταβλητή “B”. Επιστρέφει στην κατάσταση της πιο πρόσφατης μεταβλητής “A” που πήρε τιμή. Με αυτόν τον τρόπο θα αποφευχθούν οι δοκιμές στις τιμές των μεταβλητών που βρίσκονται ανάμεσα στην “A” και την “B”.



Σχήμα 11.6: Ένα παράδειγμα προβλήματος χρωματισμού.

Σημειώνεται ότι εδώ το πρόβλημα είναι αδύνατο, σκοπός της αναζήτησης είναι να το αποδείξει. Η απόδειξη αυτή θα μπορούσε να είναι μέρος ενός μεγαλύτερου αλγορίθμου, που ψάχνει να βρει ποιος είναι ο ελάχιστος αριθμός από χρώματα με τα οποία θα μπορούσαν να χρωματιστούν αυτές οι περιοχές.

Ένα παράδειγμα για την λειτουργία του Backjumping φαίνεται στο πρόβλημα του χρωματισμού. Στο πρόβλημα του χρωματισμού δίνονται περιοχές και χρώματα. Πρέπει να δοθεί από ένα χρώμα σε κάθε περιοχή έτσι ώστε δύο περιοχές με κοινά σύνορα να μην έχουν το ίδιο χρώμα. Στο σχήμα 1.6 δίνονται πέντε περιοχές και δύο χρώματα (κόκκινο, πράσινο). Οι 1,2,5 συνορεύουν μεταξύ τους, ενώ οι 3 και 4 δεν συνορεύουν με κανέναν. Αν αλγόριθμος έδινε τιμές στις μεταβλητές με την αριθμητική σειρά τους, αρχικά θα έπαιρναν χρώμα οι 1 και 2 (διαφορετικό μεταξύ τους) και μετά οι 3 και 4. Σε αυτό το σημείο ο αλγόριθμος θα έφτανε στην 5 και θα ανακάλυπτε το αδιέξοδο. Αν σε αυτό το σημείο έκανε Backtracking θα έπρεπε πρώτα να ελέγξει τις υπόλοιπες διαθέσιμες τιμές για την 3 και την 4 πριν να φτάσει στην 1 και 2 για να δοκιμάσει κάποιον άλλο συνδυασμό. Οι έλεγχοι στις 3 και την 4 δεν επηρεάζουν το τελικό αποτέλεσμα άρα θα μπορούσαν να αποφευχθούν. Αυτό πετυχαίνει η μέθοδος Backjumping επιστρέφοντας από την 5 απευθείας στην 2.

Περαιτέρω βελτιώσεις στην οπισθοδρόμηση είναι οι μέθοδοι Backchecking και Backmarking [13]. Η Backchecking θυμάται ζευγάρια μεταβλητών-τιμών των οποίων ο συνδυασμός παραβιάζει κάποιον περιορισμό και αποφεύγει να ξαναφτιάξει τον συνδυασμό αυτό ξανά. Η Backmarking θυμάται ζευγάρια μεταβλητών-τιμών που ελέγχθηκαν και είναι σωστά και αποφεύγει να τα ξαναελέγξει μετά από οπισθοχώρηση. Αν από αδιέξοδο στην μεταβλητή “B” επιστρέψει στην μεταβλητή “A”, τότε στην πορεία του ξανά προς την “B” δεν χρειάζεται να ελέγξει για ισχύς των περιορισμών που περιέχουν μεταβλητές που έχουν πάρει τιμή πριν την “A”, καθώς οι τιμές τους δεν έχουν αλλάξει. Οι Backchecking και Backmarking είναι και οι δύο μέθοδοι που προσπαθούν να αποφύγουν ελέγχους περιορισμών που έχουν ξαναγίνει. Αν θεωρηθεί ότι το μεγαλύτερο κομμάτι του ελέγχου μιας κατάστασης είναι ο έλεγχος για την εγκυρότητα των περιορισμών αυτή είναι μια καλή βελτίωση του Backjumping.

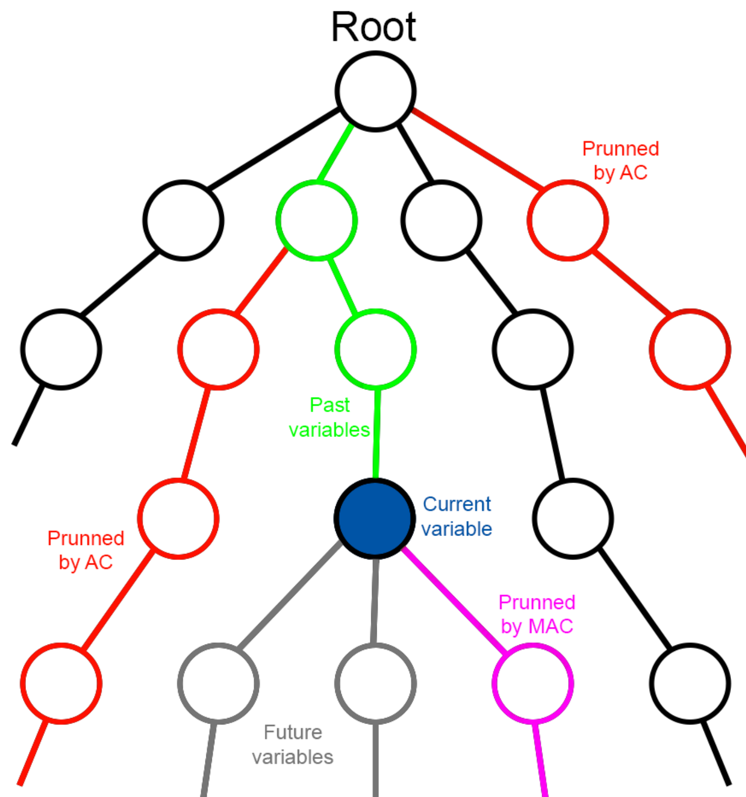
1.3.4 Προώθηση Περιορισμών

Οι μέθοδοι οπισθοχώρησης είναι καλές για να διορθώνονται τα αδιέξοδα που μπορεί να συναντήσει η αναζήτηση, αλλά δεν έχουν την δυνατότητα να τα αποφύγουν. Αυτή την δυνατότητα προσφέρουν οι μέθοδοι Προώθησης Περιορισμών (CP - Constraint Propagation). Οι CP φροντίζουν έτσι ώστε να αποφεύγονται τα αδιέξοδα σε επόμενα βήματα του αλγορίθμου προσπαθώντας να αφαιρέσουν όσες πιο πολλές προβληματικές τιμές (που θα καταλήξουν σε αδιέξοδο) από μελλοντικές μεταβλητές.

Η μέθοδος Forward Checking (FC) [26] για κάθε μεταβλητή στην οποία δίνει τιμή ελέγχει ότι οι όλες τιμές όλων των μεταβλητών που δεν έχουν πάρει τιμή ακόμα είναι συνεπείς με την νέα τιμή. Το πετυχαίνει αυτό τρέχοντας τον αλγόριθμο Συνέπειας Τόξου (AC). Χρησιμοποιώντας τον AC αφαιρεί όσες ασυνεπείς τιμές βρει από τα πεδία ορισμού των επόμενων μεταβλητών, αποφεύγοντας, κατά συνέπεια, αυτά τα αδιέξοδα. Οι τιμές πρέπει να αφαιρεθούν προσωρινά. Αυτό σημαίνει πως αν ο αλγόριθμος, παρά την προσπάθεια της AC, συναντήσει αδιέξοδο και οπισθοχωρήσει σε κάποια προηγούμενη κατάσταση πρέπει να επαναφέρει τα πεδία ορισμού των ασυμπλήρωτων μεταβλητών στο σημείο όπου ήταν σε εκείνη την κατάσταση, πριν την διαγραφή τιμών από την AC. Πέρα από την αποφυγή κάποιων αδιεξόδων η FC μπορεί να καταλάβει ότι μια τιμή είναι προβληματική, αν η AC επιστρέψει κενό πεδίο ορισμού για κάποια μελλοντική μεταβλητή, και να προχωρήσει στην επόμενη χωρίς να εξετάσει νέες καταστάσεις. Αυτά έρχονται σε μεγαλύτερο κόστος από τις μεθόδους Backtracking αλλά σχεδόν πάντα η αποφυγή αδιεξόδων είναι πιο αποδοτική από την διόρθωση τους [4].

Μια λογική ενίσχυση του FC είναι να χρησιμοποιηθεί πιο ισχυρό είδος συνέπειας σε κάθε ανάθεση, ώστε να αφαιρεθούν παραπάνω αδιέξοδα από τις μελλοντικές μεταβλητές. Η μέθοδος PLA χρησιμοποιεί DAC (Directional Arc Consistency), ενώ οι αλγόριθμοι που κάνουν κάποιο είδος AC ονομάζονται Maintaining Arc Consistency (MAC) [26]. Η ισχύς της συνέπειας που θα χρησιμοποιηθεί προφανώς αυξάνει το κόστος της χρήσης της. Για να επιλεγεί ο πιο αποδοτικός αλγόριθμος για κάποιο μοντέλο πρέπει να προηγηθούν ανάλυση πειράματα.

Συνοψίζοντας τη αναζήτηση σε δέντρο, υπάρχουν τρία κύρια είδη τεχνικών για να βελτιωθεί η απόδοση των αλγορίθμων. Οι Τεχνικές Συνέπειας δρουν προ-επεξεργαστικά σε ένα δέντρο και αφαιρούν ασυνεπείς τιμές πριν να αρχίσει η αναζήτηση. Η ισχυρή K-συνέπεια μόνο είναι πλήρης και μπορεί να λύσει ένα πρόβλημα αλλά είναι χρονοβόρα και η αναζήτηση είναι σχεδόν πάντα καλύτερη επιλογή. Η Στρατηγική Οπισθοχώρηση βοηθάει τον αλγόριθμο να κάνει συστηματική αναζήτηση, δίνοντας του την δυνατότητα να αντιδράσει δυναμικά στα αδιέξοδα. Τέλος η Προώθηση Περιορισμών καθιστά τον αλγόριθμο ικανό να αποφεύγει αδιέξοδα πριν φτάσει σε αυτά.



Σχήμα 11.7: Σχηματικό παράδειγμα τεχνικών αναζήτησης σε δέντρο.

Στο σχήμα φαίνεται ένα αυθαίρετο δέντρο με τις καταστάσεις ενός μοντέλου, σε κάθε κατάσταση επιλέγεται μια μεταβλητή και συμπληρώνεται. Αρχικά αφαιρούνται οι καταστάσεις με κόκκινο χρώμα από την AC γιατί όλες οι τιμές της μεταβλητής είναι ασυνεπείς. Μετά αρχίζει η αναζήτηση, οι μεταβλητές που έχει εξετάσει ο αλγόριθμος φαίνονται με πράσινο. Η μεταβλητή που εξετάζεται τώρα είναι μπλε. Μελλοντικές μεταβλητές είναι όλοι οι κλώνοι που πηγάζουν από την τρέχουσα μεταβλητή. Κάποιες από τις μελλοντικές μπορούν να αφαιρεθούν από την MAC γιατί όλες οι τιμές τους είναι ασυνεπείς.

Η επιλογή των τεχνικών που θα χρησιμοποιηθούν για κάποιο πρόβλημα είναι αντικείμενο που ο επιστήμονας πρέπει να αναλύσει. Δεν υπάρχει γενική λύση για όλα τα προβλήματα, κάποιες τεχνικές λειτουργούν καλύτερα σε κάποια μοντέλα από άλλες. Ακόμα και για δύο μοντέλα με παρόμοιους περιορισμούς μπορεί ένας αλγόριθμος να εμφανίσει ασταθή συμπεριφορά. Γι’ αυτό τα προβλήματα απαιτούν πειραματισμό για να βρεθεί ένας καλός αλγόριθμος για την λύση τους.

1.4 Αναζήτηση Πλήρων Αναθέσεων

Στην οργάνωση σε μορφή δέντρου ο αλγόριθμος προσπαθεί να κινηθεί από μία ημιτελή ανάθεση των μεταβλητών σε μία πλήρη ανάθεση αποφεύγοντας, όσο γίνεται, τα αδιέξοδα. Μία ακόμα οργάνωση του χώρου αναζήτησης που εφαρμόζεται στον τομέα των CSP, είναι ο χώρος των πλήρων αναθέσεων. Σε αυτόν τον χώρο υπάρχουν μόνο οι καταστάσεις του μοντέλου που έχουν συμπληρωμένες όλες τους τις μεταβλητές. Η έννοια της γειτονιάς τώρα αλλάζει: γείτονας κάθε κόμβου είναι κάθε κατάσταση που διαφέρει σε ακριβώς μία μεταβλητή και δεν υπάρχουν γονείς και παιδιά. Δύο γειτονικές καταστάσεις ενώνονται με μία ακμή, που είναι το βήμα μεταξύ τους. Ο αλγόριθμος, ξεκινώντας από

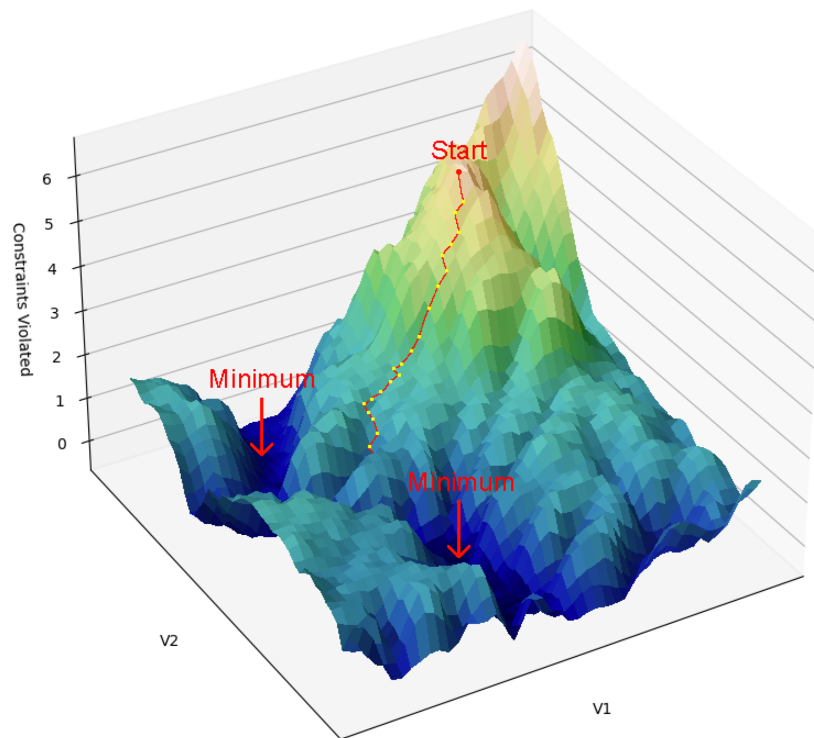
μία τυχαία κατάσταση, επιχειρεί να κινηθεί προς κάποια κατάσταση που δεν παραβιάζει περιορισμούς.

Αυτή η αλλαγή οργάνωσης του χώρου μειώνει όλες τις πιθανές καταστάσεις που θα έπρεπε να εξετάσει ένας αλγόριθμος, αφού πλέον απομένει μόνο το υποσύνολο των καταστάσεων που αντιστοιχούν στα φύλλα του δέντρου. Το γεγονός αυτό όμως κάνει πιο δύσκολη την περιήγηση του αλγορίθμου στον χώρο αυτό. Οι αλγόριθμοι που κάνουν αναζήτηση στον χώρο των πλήρων αναθέσεων πρέπει να χρησιμοποιήσουν έξυπνες τεχνικές για να αποφύγουν ένα διαφορετικό είδους αδιέξοδο: τα τοπικά ελάχιστα, που εξηγούνται στην επόμενη ενότητα.

1.4.1 Τοπική Αναζήτηση

Η αναζήτηση σε ένα γράφημα που δεν έχει την μορφή δέντρου λέγεται τοπική αναζήτηση [33]. Η τοπική αναζήτηση είναι μια μέθοδος που έχει εφαρμογές σε προβλήματα βελτιστοποίησης. Σε τέτοια προβλήματα ο χώρος διαμορφώνεται από τις τιμές μιας αντικειμενικής συνάρτησης. Στόχος είναι να βρεθεί η είσοδος που καταλήγει στην ελάχιστη τιμή. Στα CSP, σε γενικές γραμμές, η είσοδος της συνάρτησης είναι η κατάσταση του μοντέλου με όλες τις μεταβλητές συμπληρωμένες και αντικειμενική συνάρτηση είναι ο αριθμός των περιορισμών που παραβιάζονται. Ο λόγος που η αναζήτηση είναι τοπική είναι γιατί ο αλγόριθμος δεν θα μπορούσε να υπολογίσει την αντικειμενική συνάρτηση για κάθε πιθανή πλήρη κατάσταση και να τις ταξινομήσει. Οι καταστάσεις αυτές όπως και στο δέντρο είναι πάρα πολλές. Έτσι ο αλγόριθμος έχει πάντα μια μικρή εικόνα του χώρου και προσπαθεί να κινηθεί σε μια κατεύθυνση που θα τον οδηγήσει προς την λύση. Για να μπορέσει να βρει αυτή την κατεύθυνση ο αλγόριθμος χρησιμοποιεί μια ευρετική συνάρτηση.

Ο αλγόριθμος Hill Climbing (HC) [19] είναι ο πιο διάσημος αλγόριθμος τοπικής αναζήτησης για CSP. Ξεκινώντας από μια τυχαία κατάσταση στον χώρο πλήρων αναθέσεων, κάνει βήμα (αλλάζοντας την τιμή σε μία μεταβλητή) προς την γειτονική κατάσταση που παραβιάζει τους λιγότερους περιορισμούς. Εδώ φαίνεται αμέσως το κύριο πρόβλημα της τοπικής αναζήτησης: αν τυχαίνει καμία από τις καταστάσεις τη γειτονιάς να έχει καλύτερη αντικειμενική τιμή (να παραβιάζει δηλαδή λιγότερους περιορισμούς) από την τρέχουσα τότε κανένα από τα πιθανά βήματα δεν συμφέρει. Αυτό σημαίνει πως ο αλγόριθμος έχει φτάσει σε ένα τοπικό ελάχιστο και έχει κολλήσει. Σε περιπτώσεις γενικής βελτιστοποίησης ίσως το τοπικό ελάχιστο αυτό να ήταν αποδεκτό, όμως στα CSP συνήθως δεν είναι, καθώς παραβιάζονται ακόμα περιορισμοί. Όταν ο HC φτάσει σε τοπικό ελάχιστο πρέπει να ξεκινήσει από μια καινούργια τυχαία κατάσταση.



Σχήμα 11.8 Γραφικό παράδειγμα χώρου τοπικής αναζήτησης.

Χρησιμοποιώντας ένα παράδειγμα μοντέλου 2 μεταβλητών (V1,V2), έτσι ώστε να μπορεί να σχεδιαστεί σε τρείς διαστάσεις το γράφημα της αντικειμενικής συνάρτησης.

Στο σχήμα 1.8 φαίνεται ο χώρος αναζήτησης και η λειτουργία του HC γραφικά. Το τρισδιάστατο γράφημα οπτικοποιεί την αντικειμενική συνάρτηση ενός μοντέλου με δύο μεταβλητές. Ο χώρος τιμών της αντικειμενικής στα CSP με διακριτά πεδία ορισμού είναι διακριτός, για να φαίνεται καλύτερα εδώ σχεδιάστηκε με συνεχή συνάρτηση. Τώρα φαίνεται οπτικά ο "λόφος" που "σκαρφαλώνει" ο HC, αν και σε αυτή την περίπτωση κάνει ελαχιστοποίηση και όχι μεγιστοποίηση (οι δύο μέθοδοι είναι συμπληρωματικές με αλλαγή προσήμου της αντικειμενικής συνάρτησης). Ξεκινάει από το κόκκινο σημείο και κάνει τα βήματα που φαίνονται στα κίτρινα σημεία και στόχος του είναι να φτάσει σε κάποιο από τα ελάχιστα. Όπως φαίνεται όμως πέφτει σε ένα τοπικό ελάχιστο και παγιδεύεται. Ο HC έχει μια ακόμα πιο περιορισμένη εικόνα αυτού του χώρου, μόνο τις γειτονιές από τα κίτρινα σημεία και αυτό εξηγεί το γεγονός ότι θα χρειαστεί πολλές επανεκκινήσεις για να βρει κάποιο ελάχιστο. Η περιορισμένη εικόνα όμως είναι και θετικό χαρακτηριστικό καθώς ο αλγόριθμος κάνει αρκετά λιγότερη δουλειά από μια πλήρη αποτίμηση της αντικειμενικής συνάρτησης.

Το σχήμα 1.8 μπορεί να κάνει την τοπική αναζήτηση να φαντάζει απλή αλλά στην πραγματικότητα τα προβλήματα έχουν πολύ μεγαλύτερα πεδία ορισμού. Το γεγονός αυτό θα έκανε δύσκολη την αποτίμηση, για οπτικοποίηση, της αντικειμενικής σε ολόκληρο τον χώρο αναζήτησης. Το πραγματικό εμπόδιο στην οπτικοποίηση είναι όμως οι διαστάσεις των προβλημάτων: σπάνια ένα πρόβλημα θα έχει μια ή δύο μεταβλητές. Για N μεταβλητές η τοπική αναζήτηση πρέπει να ψάξει σε έναν

N+1διάστατο χώρο. Παρότι το σχήμα 1.8 βοηθάει να αποκτηθεί μια γενική ιδέα της τοπικής αναζήτησης οι αλγόριθμοι συχνά έχουν να περιηγηθούν σε πολύ πιο περίπλοκους χώρους από αυτούς που μπορεί να δει ο άνθρωπος.

Μια βελτίωση του HC είναι ο Min-Conflicts (MC) [20] αντί να εξετάζει όλες τις γειτονικές καταστάσεις της τρέχουσας, επιλέγει τυχαία μια μεταβλητή μόνο από αυτές που συμμετέχουν σε περιορισμούς. Για την μεταβλητή επιλέγει την τιμή που "διορθώνει" τους περισσότερους περιορισμούς. Για να αποφύγει τα τοπικά ελάχιστα ο MC αποφεύγει να διορθώσει την ίδια μεταβλητή πάνω από μια φορά. Ο τρόπος με τον οποίο θα κολλούσε θα ήταν αν έμπαινε σε έναν βρόγχο στον οποίο θα διόρθωνε ξανά και ξανά την ίδια ακολουθία από μεταβλητές. Για να μην συμβεί αυτό ο MC κρατάει μια λίστα από τις μεταβλητές που έχει διορθώσει και αν οι μεταβλητές που παραβιάζουν κάποιο περιορισμό είναι μόνο μεταβλητές που έχουν ήδη διορθωθεί κάνει οπισθοχώρηση σε κάποια προηγούμενη κατάσταση.

Παίρνοντας συγκεκριμένα την ιδέα της τυχαίας επιλογής μεταβλητής από τον MC, είναι ξεκάθαρο πως η τυχαιότητα παίζει σημαντικό ρόλο στην αποφυγή των τοπικών ελαχίστων. Η μέθοδος Random Walk (RW) [21] έχει μια πιθανότητα ρ βάσει της οποίας επιλέγει πως θα προχωρήσει. Με πιθανότητα ρ επιλέγει τυχαία μια μεταβλητή και τις αλλάζει τυχαία την τιμή (στην [21] αναφέρεται σε προβλήματα SAT, όπου οι μεταβλητές παίρνουν τιμές μόνο 0 ή 1 άρα αντιστρέφεται απλά η τιμή), ενώ με πιθανότητα 1-ρ εφαρμόζει κανονικά την HC ή MC. Αυτή η μικρή τροποποίηση της τοπικής αναζήτησης αρκεί για μην κολλάει σε ελάχιστα και να βελτιωθεί η απόδοση της.

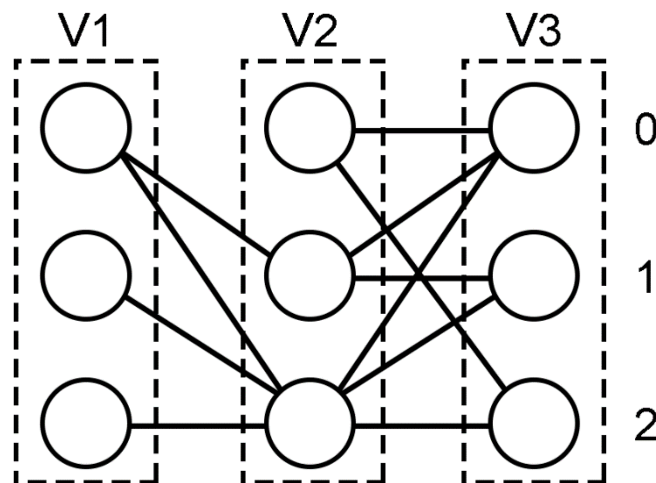
Η μέθοδος της Προσομοιωμένης Ανόπτησης (SA - Simulated Annealing) [22] παίρνει την έμπνευση της από την φυσική. Όταν αυξάνεται η θερμοκρασία ενός υλικού τα άτομα του αρχίζουν να κινούνται περισσότερο και η σύσταση του να γίνεται πιο ρευστή. Η SA χρησιμοποιεί την ιδέα αυτή επιλέγοντας την επόμενη κατάσταση με βάση την θερμοκρασία, που είναι τώρα μια μεταβλητή που ελέγχει τον αλγόριθμο. Όλες οι πιθανές κινήσεις παίρνουν μία πιθανότητα να συμβούν, αν η θερμοκρασία χαμηλή τότε οι καλές κινήσεις (αυτές που μειώνουν περισσότερο την αντικειμενική συνάρτηση) έχουν μεγαλύτερη πιθανότητα να συμβούν. Αν η θερμοκρασία είναι υψηλή τότε έχουν όλες οι κινήσεις είναι σχεδόν το ίδιο πιθανές. Ο έλεγχος της θερμοκρασίας κατά την εκτέλεση του αλγόριθμου είναι σημαντικός. Η θερμοκρασία συνήθως ξεκινάει από μεγάλη τιμή και πέφτει. Αυτό επιτρέπει πιο τυχαίες κινήσεις στην αρχή και σταθεροποιεί τον αλγόριθμο στο τέλος, όταν φτάνει κοντά στην λύση. Ο ρυθμός μείωσης της θερμοκρασίας είναι σημαντικός διότι ο αλγόριθμος δεν θα φτάσει σε καλή λύση αν "κρυώσει" πολύ γρήγορα ή πολύ αργά.

Η τεχνική της Αναζήτησης Ταμπού (TS - Tabu Search) [23] δεν βασίζεται στην τυχαιότητα αλλά εισάγει την ιδέα της μνήμης ταμπού. Κατά την διάρκεια της αναζήτησης ο αλγόριθμος μπορεί να έχει μια ή παραπάνω λίστες ταμπού στις οποίες μπορεί να εισάγει ή να αφαιρέσει καταστάσεις ή χαρακτηριστικά καταστάσεων. Η χρησιμότητα των λιστών αυτών βρίσκεται στο ότι ο αλγόριθμος μπορεί να θυμάται προηγούμενες καταστάσεις οι οποίες είναι καλές ή κακές. Έτσι μπορεί να αποφεύγει τους βρόγχους και να μην εγκλωβίζεται σε τοπικά ελάχιστα. Το μέγεθος των λιστών είναι σημαντικό καθώς όταν μια λίστα γεμίσει το παλαιότερο στοιχείο αφαιρείται για να εισαχθεί το

καινούριο, με μέθοδο FIFO (First In First Out). Άρα το μέγεθος της κάθε λίστας αντιπροσωπεύει το μέγεθος της μνήμης του αλγορίθμου.

Πέρα από το να αποφεύγει προηγούμενες καταστάσεις, ο αλγόριθμος TS μπορεί να χρησιμοποιήσει κάποια λίστα για να θυμάται καλά χαρακτηριστικά από κάποιες προηγούμενες καταστάσεις. Τα χαρακτηριστικά αυτά είναι αναθέσεις τιμών και μπορούν να αποθηκευτούν μαζί με τον αριθμό περιορισμών που παραβιάζουν. Άρα ο TS μπορεί για παράδειγμα, σε ένα μοντέλο με μεταβλητές (V1,V2,V3,V4,V5) να αποθηκεύσει σε μια από τις λίστες ταμπού του, ότι μια συγκεκριμένη ανάθεση A των μεταβλητών (V1,V2,V3) δεν παραβιάζει περιορισμούς. Μετά αν βρει μια ανάθεση B των (V4,V5) που επίσης δεν παραβιάζει περιορισμούς να την αποθηκεύσει και αυτή. Η λίστα αυτή αποθηκεύει καλά χαρακτηριστικά, αντίστοιχα μπορεί να υπάρχει μια λίστα που να αποθηκεύει κακά χαρακτηριστικά. Όπως και με το να αποθηκεύει καταστάσεις ολόκληρες, ο TS θα έχει περιορισμούς που θα του απαγορεύουν να επισκεφτεί καταστάσεις που δεν έχουν αυτά τα καλά χαρακτηριστικά και θα του απαγορεύουν να επισκεφτεί καταστάσεις που έχουν τα κακά.

Τα αποθηκευμένα χαρακτηριστικά επίσης βοηθούν στην δημιουργία ενός νέου γείτονα για την τρέχουσα κατάσταση. Ο TS δηλαδή θα μπορούσε να συνδυάσει τα A και B και να φτιάξει μια καινούργια κατάσταση με αυτές τις αναθέσεις, αντί να κάνει βήμα σε κάποιον από τους τρέχοντες γείτονες, μπορεί να επιλέξει να κάνει άλμα σε μια από τις καταστάσεις που μπορεί να δημιουργήσει από την μνήμη χαρακτηριστικών. Ακόμη ο TS μπορεί να επιλέξει να αφαιρέσει κάποια κατάσταση/χαρακτηριστικό από μια λίστα ανάλογα με κριτήρια φιλοδοξίας (aspiration criteria) καθώς και να προσθέσει τυχαιότητα στην λειτουργία του [23]. Είναι ένας αλγόριθμος που δίνει πολύ χώρο για παραμετροποίηση και δυναμική εξέλιξη, κατά την εκτέλεση του, έτσι ώστε να ταιριάξει στις ανάγκες του προβλήματος.



Σχήμα 11.9: Παράδειγμα νευρωνικού GENET.

Το μοντέλο έχει τρεις μεταβλητές (V1,V2,V3) με πεδία ορισμού (0,1,2) και οι τρείς. Στο δίκτυο υπάρχουν εννιά κόμβοι, ένας για κάθε πιθανή τιμή κάθε μεταβλητής. Οι περιορισμοί που υλοποιούνται από τις συνάψεις που

φαίνονται είναι οι “V1 ≤ V2” και “V3 = V2+1”. Για παράδειγμα ο κόμβος “V1=0” συνδέεται με τους “V2=1” και “V2=2” γιατί τα δύο ζευγάρια αναθέσεων παραβιάζουν τον πρώτο περιορισμό.

Ο αλγόριθμος GENET [27] χρησιμοποιεί την μορφή νευρωνικού δικτύου για να λύσει τα CSP. Οι αλγόριθμοι τοπικής αναζήτησης, ιδιαίτερα αυτοί οι οποίοι βασίζονται σε τυχαίες κινήσεις/επανεκκινήσεις για να αποφύγουν τα τοπικά ελάχιστα, έχουν κακή απόδοση σε χώρους με πολλά τοπικά και λίγα ολικά ελάχιστα. Γι’ αυτό τον λόγο ο GENET κάνει αναζήτηση στον χώρο πλήρων αναθέσεων προσομοιώνοντας ένα νευρωνικό δίκτυο. Σε αυτή την περίπτωση ο όρος “νευρωνικό δίκτυο” δεν αναφέρεται στα πιο γνωστά Multilayer Perceptrons (MLP) [28] αλλά σε ένα διαφορετικό δίκτυο από νευρώνες.

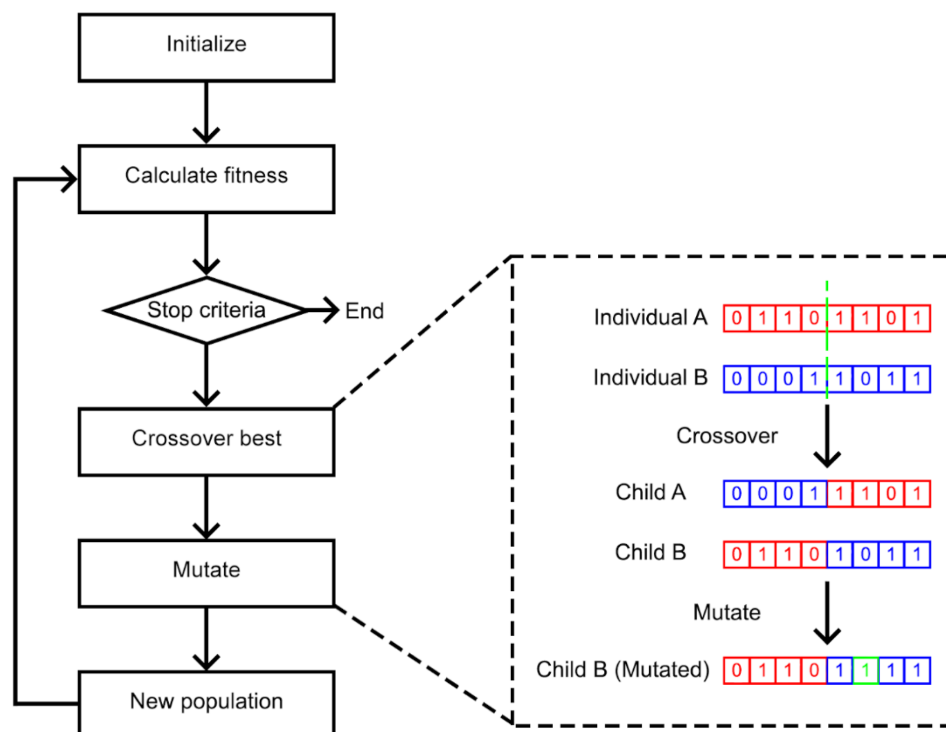
Το δίκτυο του GENET φτιάχνεται με νευρώνες όλες τις πιθανές αναθέσεις τιμών σε μεταβλητές. Οι νευρώνες μπορούν να είναι είτε ενεργοί (τιμή 1) ή ανενεργοί (τιμή 0). Συνάψεις ανάμεσα στους νευρώνες είναι οι περιορισμοί και υπάρχουν μόνο μεταξύ των τιμών δύο διαφορετικών μεταβλητών. Συγκεκριμένα δύο νευρώνες συνδέονται μόνο όταν ο συνδυασμός των τιμών τους απαγορεύεται από κάποιον περιορισμό. Ένα παράδειγμα νευρωνικού GENET φαίνεται στο σχήμα 1.9. Όπως και στα MLP οι συνάψεις έχουν βάρος, στο GENET τα βάρη είναι πάντα αρνητικά και αρχικοποιούνται στο -1. Μέσα από συνεχείς ανανεώσεις το GENET μαθαίνει τα βάρη που το οδηγούν στη λύση.

Η λειτουργία του GENET, χωρίζει τους κόμβους σε ομάδες, κάθε ομάδα έχει τις τιμές μιας μόνο μεταβλητής (συμβολίζονται με διακεκομμένες γραμμές στο σχήμα 1.9). Ξεκινάει ενεργοποιώντας τυχαία έναν νευρώνα από κάθε ομάδα, αυτή είναι η αρχική τυχαία κατάσταση στον χώρο πλήρων αναθέσεων. Σε κάθε επανάληψη ένας νευρώνας δέχεται σαν είσοδο το άθροισμα των καταστάσεων των συνδεδεμένων σε αυτόν νευρώνες πολλαπλασιασμένων με το βάρος της σύναψης. Μόνο ο νευρώνας με την μεγαλύτερη είσοδο ενεργοποιείται σε μια ομάδα (η τιμή του γίνεται 1), αυτή είναι η τιμή που επιλέχθηκε γι’ αυτή την μεταβλητή σε αυτή την επανάληψη. Σε περιπτώσεις ίσης ελάχιστης εισόδου ανάμεσα σε δύο η παραπάνω επιλέγεται ένας τυχαία. Όταν σε όλες τις ομάδες ο ενεργοποιημένος νευρώνας έχει μηδενική είσοδο τότε δεν παραβιάζεται κανένας περιορισμός και το δίκτυο έχει συγκλίνει στην λύση.

Αν σε κάποια επανάληψη ένας τουλάχιστον ενεργοποιημένος νευρώνας έχει αρνητική είσοδο και το δίκτυο δεν κάνει αλλαγές τότε έχει κολλήσει σε τοπικό ελάχιστο, καθώς παραβιάζεται ένας τουλάχιστον περιορισμός. Τότε το GENET επιλέγει να ανανεώσει το βάρος της κάθε σύναψης, ανάμεσα στους κόμβους i και j στο δίκτυο προσθέτοντας $\Delta W_{i,j} = -s_i \times s_j$ στο βάρος της, όπου s_i είναι η κατάσταση ενεργοποίησης του νευρώνα i. Σε αντίθεση με μεθόδους τυχαίας κατεύθυνσης που χρησιμοποιούνται συνήθως, αυτή η μέθοδος έχει μεγαλύτερη βελτίωση της απόδοσης του GENET [27].

Εξετάζοντας περεταίρω την ιδέα του συνδυασμού χαρακτηριστικών του TS, μπορεί να επεκταθεί σαν Γενετικός Αλγόριθμος (GA - Genetic Algorithm) [29]. Οι GA είναι αλγόριθμοι που γενικότερα χρησιμοποιούνται στον τομέα της βελτιστοποίησης. Ο αλγόριθμος έχει έναν πληθυσμό από άτομα

(individuals), το κάθε άτομο έχει ένα χρωμόσωμα (chromosome) το οποίο αντιστοιχεί σε κάποιο σημείο του χώρου αναζήτησης. Μια συνάρτηση καταλληλότητας (fitness) χρησιμοποιείται για να δώσει μια τιμή καταλληλότητας στο κάθε άτομο ανάλογα με το πόσο καλό είναι το χρωμόσωμα του. Αυτή είναι η αντικειμενική συνάρτηση για τους GA. Ανάλογα με την σχεδίαση του αλγορίθμου δύο ή παραπάνω από τα καλύτερα άτομα επιλέγονται, σε κάθε επανάληψη, και τα χρωμοσώματα τους συνδυάζονται (crossover) και έπειτα μεταλλάσσονται (mutate) με κάποια πιθανότητα, για να φτιάξουν νέα άτομα για τον πληθυσμό της επόμενης επανάληψης. Τα άτομα με μικρότερες τιμές καταλληλότητας αποβάλλονται. Όταν, σε κάποια επανάληψη, ένα από τα άτομα είναι αρκετά καλό έτσι ώστε το χρωμόσωμα του να θεωρείται ολικό βέλτιστο η αναζήτηση σταματά. Η μέθοδος βασίζεται στην θεωρία της εξέλιξης και πετυχαίνει στο να συγκλίνει σε κάποια λύση σε πολλά προβλήματα.



Σχήμα 11.10: Σχεδιάγραμμα των γενετικών αλγορίθμων.

Η διαδικασία του συνδυασμού συνήθως περιλαμβάνει την τομή των χρωμοσωμάτων, σε κάποιο σημείο κατά μήκος, δύο ατόμων και την επανακόλληση τους με το κομμάτι του άλλου χρωμοσώματος. Η μετάλλαξη επιλέγει ένα από τα κομμάτια του χρωμοσώματος και τα αλλάζει, συμβαίνει με μικρή πιθανότητα σε κάποια παιδιά.

Επειδή τα μοντέλα CSP δεν έχουν κάποια αντικειμενική συνάρτηση και οι GA δεν έχουν κάποιο τρόπο να χειρίζονται περιορισμούς, πρέπει να γίνει προσαρμογή τους για να λυθούν με γενετικούς αλγόριθμους. Υπάρχουν δύο τρόποι να χειριστεί ένας GA τους περιορισμούς ενός προβλήματος, ο άμεσος και ο έμμεσος [36,37]. Στον άμεσο χειρισμό περιλαμβάνονται μέθοδοι συνδυασμού οι οποίοι να διατηρούν την λύση του εφικτή, εφόσον οι γονείς έχουν εφικτή λύση ή επιδιορθώνουν την λύση του παιδιού. Οι μέθοδοι αυτές είναι πιο δύσκολες στην υλοποίηση γιατί εξαρτώνται από το μοντέλο.

Στον έμμεσο χειρισμό οι περιορισμοί μπαίνουν στην συνάρτηση καταλληλότητας και δίνουν ποινές στην λύση του ατόμου ανάλογα με τον αριθμό των περιορισμών που παραβιάζει.

O GAVI (Genetic Algorithm with Viral Infection) είναι ένας γενικός αλγόριθμος που μπορεί χρησιμοποιηθεί για λύση CSP [30,31]. Αρχικά γίνεται αναζήτηση σε όλο τον χώρο χρησιμοποιώντας τον GA και έπειτα επιλέγονται τα καλύτεροι άτομα (elites) του πληθυσμού με την συνάρτηση καταλληλότητας. Όσοι από τους elites έχουν τιμή καταλληλότητας μεγαλύτερη από ένα όριο βελτιώνονται χρησιμοποιώντας MC. Μετά την τοπική αναζήτηση με MC, οι καλύτεροι του πληθυσμού επιλέγονται και συνδυάζονται, αν τυχαίνει κάποιος elite να έχει κολλήσει σε τοπικό ελάχιστο τότε αφαιρείται από τον πληθυσμό. Από τα άτομα που δεν ανήκουν στους elites, κάποια από αυτά αφαιρούνται ανάλογα με την απόστασή τους από τους elites. O αλγόριθμος σταματά όταν κάποιο από τα άτομα φτάσει σε κατάσταση που αποτελεί λύση του μοντέλου.

1.5 Προβλήματα Ικανοποίησης - Βελτιστοποίησης Περιορισμών

Σε αρκετές περιπτώσεις πραγματικών προβλημάτων δεν αρκεί να βρεθεί μια εφικτή λύση: απαιτείται επιπλέον η λύση αυτή να είναι βέλτιστη. Προβλήματα με περιορισμούς στα οποία η ποιότητα της λύσης είναι σημαντική λέγονται Προβλήματα Ικανοποίησης-Βελτιστοποίησης Περιορισμών (CSOP - Constraint Satisfaction Optimization Problems) [4]. Η ποιότητα της λύσης μπορεί να μετρηθεί με μια αντικειμενική συνάρτηση και ο στόχος είναι να βρεθεί μια λύση που να την βελτιστοποιεί.

Ο τύπος αλγορίθμου που λύνει CSOP είναι ο Branch and Bound (B&B) [34]. Η ιδέα του B&B είναι ότι διχοτομεί έναν τετραγωνικό χώρο, μιας συνάρτησης, σε δύο ίσα μικρότερα κομμάτια. Διαλέγει το κομμάτι που έχει τα καλύτερα άνω και κάτω όρια (upper and lower bound) και επαναλαμβάνει διχοτομώντας αυτό. Για να υπολογίσει τα όρια χρησιμοποιεί ευρετικές συναρτήσεις. Ο Depth First Branch and Bound (DFBB) [2] υπολογίζει το κάτω όριο της αντικειμενικής συνάρτησης ενώ κάνει αναζήτηση στο δέντρο καταστάσεων ενός μοντέλου CSOP. Έχοντας πάρει ένα άνω όριο εξ αρχής, που είναι το μέγιστο όριο που θα δεχτεί ο αλγόριθμος, μπορεί να κλαδέψει μια κατάσταση και όλους τους απογόνους της αν *άνω όριο* ≤ *κάτω όριο*. Πέρα από το κλάδεμα ο DFBB δρα σαν Backtracking και η απόδοση του εξαρτάται από το άνω όριο και την ισχύ της ευρετικής που βρίσκει το κάτω όριο [4].

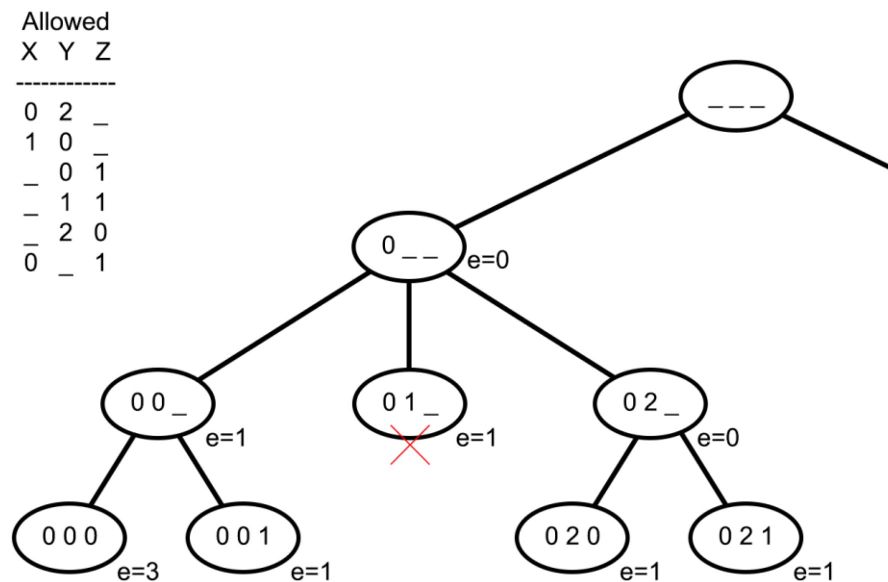
Οι γενετικοί αλγόριθμοι που συζητήθηκαν στο κεφάλαιο της Αναζήτησης Πλήρων Αναθέσεων μπορούν επίσης να χρησιμοποιηθούν για προβλήματα CSOP. Αρκεί η να προστεθεί μια αντικειμενική συνάρτηση ποιότητας λύσης σε μεθόδους άμεσου χειρισμού περιορισμών ή η προσθήκη της αντικειμενική συνάρτηση ποιότητας στην αντικειμενική συνάρτηση παραβίασης περιορισμών.

1.6 Προβλήματα Ελαστικών Περιορισμών

Τα Προβλήματα Ελαστικών Περιορισμών (SCSP - Soft Constraint Satisfaction Problems) έχουν μεγάλη ομοιότητα με τα CSOP [4]. Τώρα αντί για να ψάχνουν την βέλτιστη λύση ψάχνουν μια λύση που παραβιάζει τους λιγότερους περιορισμούς. Αυτή είναι η βέλτιστη λύση ή οποία δεν είναι πλήρης. Ο λόγος που μπορεί να είναι αποδεκτή μια μη-πλήρης λύση είναι είτε γιατί το πρόβλημα είναι υπέρ-περιορισμένο (overconstrained) και καμία λύση δεν είναι εφικτή είτε γιατί μπορεί μια λύση να πρέπει

να επιτευχθεί σε περιορισμένο χρονικό διάστημα και το πρόβλημα είναι πολύ δύσκολο [35]. Για να βρεθεί αυτή η βέλτιστη λύση η αντικειμενική συνάρτηση μιας κατάστασης γίνεται ο αριθμός των περιορισμών που παραβιάζει.

Όπως είχε συζητηθεί στο κεφάλαιο της τοπικής αναζήτησης, οι μέθοδοι τοπικής αναζήτησης ήδη χρησιμοποιούν την αντικειμενική συνάρτηση αυτή για να εκτιμήσουν την ποιότητα της λύσης που έχουν βρει. Οπότε είναι λογικό ότι μπορούν να υιοθετηθούν και για την λύση SCSP, αρκεί να σταματήσουν όταν ο αριθμός των παραβιασμένων περιορισμών είναι αποδεκτός ή περάσει ο επιτρεπτός χρόνος αναζήτησης.



Σχήμα 11.11: Παράδειγμα λειτουργίας της P-BB.

Εδώ φαίνεται η λειτουργία της P-BB σε ένα μοντέλο με μεταβλητές (X,Y,Z) και πεδία ορισμού Dx=(0,1), Dy=(0,1,2), Dz=(0,1). Οι δυαδικοί περιορισμοί Cxy,Cxz,Cyz φαίνονται στο σχήμα. Η μέθοδος ξεκινά από την αρχική κατάσταση έχοντας ένα μεγάλο άνω όριο παραβιάσεων N. Σε κάθε κόμβο φαίνονται οι παραβιάσεις/σφάλματα e. Όταν φτάσει στο πρώτο φύλλο “0 0 0” βλέπει ότι αυτή η πλήρης ανάθεση έχει 3 σφάλματα, παραβιάζει και τους τρεις περιορισμούς. Άρα θέτει το N=3. Στο επόμενο φύλλο “0 0 1” παραβιάζεται μόνο ο Cxy άρα θέτει νέο άνω όριο N=1. Όταν συναντά την κατάσταση “0 1 _” αυτή έχει ήδη e=1 άρα δεν γίνεται να την διαδέχεται φύλλο με καλύτερη λύση από την “0 0 1” και την κλαδεύει. Στην “0 2 _” συνεχίζει κανονικά γιατί e<N.

Η μέθοδος B&B για βελτιστοποίηση CSP μπορεί να εφαρμοστεί και σε SCSP και ο συνδυασμός της με τις τεχνικές της οπισθοχώρησης και προώθησης περιορισμών μπορεί να αποβεί αρκετά αποδοτικός [35]. Χρησιμοποιώντας απλό Backtracking με B&B (P-BB - Partial Branch and Bound), ενώ το backtracking προσπαθεί να βρει μια αποδεκτή λύση ο B&B θυμάται την καλύτερη λύση που έχει δει και κλαδεύει κόμβους όταν καταλαβαίνει ότι δεν γίνεται να τους διαδέχονται φύλλα με πιο καλή

λύση. Η αξία μιας λύσης είναι ο αριθμός των περιορισμών που παραβιάζει, μια αρχική τιμή μέγιστων παραβιασμένων περιορισμών μπορεί να δοθεί για να αποτραπεί ο αλγόριθμος από το να ψάχνει μια μερική λύση που είναι ήδη μη αποδεκτή. Όταν προστίθεται και Backjumping, ο P-BJ (Partial Backjumping), μπορεί να κλαδεύει κόμβους όταν ανακαλύψει ότι μια τιμή μιας μεταβλητής δεν μπορεί να συνδυαστεί με καμία μιας άλλης. Για παράδειγμα αν βρίσκεται στο όριο των επιτρεπτών παραβιάσεων και επιλεγεί η τιμή x_a για την μεταβλητή A και, παρακάτω στο δέντρο, η τιμή x_b για την μεταβλητή B, αν η x_b με την x_a παραβιάζουν κάποιον περιορισμό ο P-BJ δεν κάνει backjump στην A, όπως θα έκανε κανονικά. Αντί γι' αυτό περιμένει να δοκιμάσει όλες τις τιμές της B και αν καμία δεν ταιριάζει με την x_a τότε κλαδεύει ολόκληρο τον κόμβο όπου A = x_a, χωρίς να δοκιμάσει άλλες τιμές για τις ενδιάμεσες μεταβλητές. Η P-BMK (Partial Backmarking), όπως και η απλή Backmarking, προσπαθεί να αποφύγει επανελέγχους περιορισμών. Θυμάται για κάθε τιμή x_v μιας μεταβλητής V σε πιο επίπεδο παραβίασε έναν περιορισμό, αν το backjump που γίνει είναι πριν από αυτό το επίπεδο τότε η τιμή που δόθηκε στην μεταβλητή εκείνου του επιπέδου δεν έχει αλλάξει και η x_v ακόμα θα παραβιάζει τον περιορισμό. Αν το backjump πάει στο επίπεδο που παραβιάστηκε ο περιορισμός ή παραπάνω πρέπει η x_v να ξανά ελεγχθεί.

Όπως και με τις τεχνικές Look Back ο B&B μπορεί να συνδυαστεί και με τις τεχνικές Look Forward [35]. Χρησιμοποιώντας τον αλγόριθμο AC, πριν την αρχή της αναζήτησης, αντί για να αφαιρεθούν οι ασυνεπείς τιμές μπορεί να αποθηκευτεί ο αριθμός συνεπειών για μια τιμή. Αυτός ο αριθμός αντιπροσωπεύει με πόσες μεταβλητές συγκρούεται αυτή η τιμή, αρκεί να είναι ασυνεπής με το πεδίο ορισμού τους. Οι τιμές του P-ACC (Partial Arc Consistency Check) μπορούν να χρησιμοποιηθούν με δύο τρόπους. Αν είναι γνωστό ένα ελάχιστο όριο παραβιασμένων περιορισμών τότε μπορούν να αφαιρεθούν οι τιμές που έχουν αριθμό συγκρούσεων πάνω από το όριο. Κατά την διάρκεια της αναζήτησης αν το ελάχιστο όριο (η λύση με τους λιγότερους παραβιασμένους περιορισμούς) είναι μικρότερο από τον αριθμό συγκρούσεων μιας τιμής τότε αυτή μπορεί να κλαδευτεί από το δέντρο. Η απόδοση αυτής της μεθόδου μπορεί να βελτιωθεί αν χρησιμοποιηθεί με την μορφή FC. Οι αλγόριθμοι P-FC (Partial Forward Checking) 1 έως 3 [35] ελέγχουν το AC των επόμενων μεταβλητών και προσθέτουν τις συνέπειες στο άθροισμα τους. Αν μια τιμή θα ξεπερνά το κάτω όριο παραβιάσεων κλαδεύεται από το δέντρο.

Πίνακας 11.1: Οι αλγόριθμοι τοπικής αναζήτησης και οι εφαρμογές τους.

Algorithm	CSP	CSOP	SCSP
Hill Climb [19]	✓	✗	✗
Min Conflicts [20]	✓	✗	✗
Random Walk [21]	✓	✗	✗
Tabu Search [23]	✓	✗	✓

Simulated Annealing [22]	✓	✗	✓
GENET [27]	✓	✗	✗
GAVI [30,31]	✓	✓	✓
B&B [34]	✗	✓	✓

Σε κάποιες εφαρμογές γίνεται να σχεδιαστεί μια ιεραρχία περιορισμών [44]. Η ιεραρχία περιλαμβάνει τους περιορισμούς σε n ομάδες H_i όπου το i συμβολίζει την προτεραιότητα, από 0 που είναι οι υποχρεωτικοί περιορισμοί έως n που είναι οι πιο αδύναμοι. Για δύο πλήρεις αναθέσεις A και B ενός μοντέλου με ιεραρχία περιορισμών H, δίνονται οι συναρτήσεις σύγκρισης *weighted-sum-better(A,B,H), worst-case-better(A,B,H)* και *least-squares-better(A,B,H).* Οι συναρτήσεις συγκρίνουν τις αναθέσεις περιορισμούς βάσει των βαρών που δίνονται σε κάθε επίπεδο της H και ανάλογα με τον αριθμό των περιορισμών που παραβιάζονται σε κάθε επίπεδο. Η ιεραρχία περιορισμών μπορεί να χρησιμοποιηθεί με δύο προσεγγίσεις [44]. Στην προσέγγιση διύλισης (refinement) οι τιμές των μεταβλητών ξεκινούν από τα αρχικά πεδία ορισμού και μειώνονται καθώς προστίθενται ένας-ένας οι περιορισμοί. Στην προσέγγιση της διατάραξης (perturbation) το μοντέλο ξεκινάει από μια λυμένη κατάσταση και αφού αλλάξουν τιμή μερικές μεταβλητές ο αλγόριθμος πρέπει να το φέρει πάλι σε αποδεκτή λύση με το ελάχιστο σφάλμα στους ελαστικούς περιορισμούς.

1.1 Συμπεράσματα

Σε αυτό το κεφάλαιο δόθηκε μια πλήρης παρουσίαση των μεθόδων και τεχνικών για την επίλυση Προβλημάτων Ικανοποίησης Περιορισμών. Ένα πρόβλημα του πραγματικού κόσμου μοντελοποιείται με μεταβλητές, πεδία ορισμού για τις μεταβλητές και περιορισμούς που περιορίζουν τις τιμές των μεταβλητών. Μετά μέσω του Προγραμματισμού Ικανοποίησης περιορισμών οργανώνεται η αναζήτηση μιας λύσης για το μοντέλο. Επειδή οι καταστάσεις που συνήθως έχουν τα μοντέλα είναι πολλές το πρόβλημα της αναζήτησης είναι NP-Complete, που σημαίνει ότι δεν υπάρχει αλγόριθμος που να βρίσκει λύση σε πολυωνυμικό χρόνο. Οι μέθοδοι αναζήτησης της λύσης είναι πολλές και παραμετροποιήσημες, μπορούν να χωριστούν σε δύο κατηγορίες: Αναζήτηση σε Δέντρο και Τοπική Αναζήτηση. Δεν υπάρχει μια γενική που να λύνει όλα τα μοντέλα. Διαφορετικά πραγματικά προβλήματα έχουν διαφορετικές ανάγκες και η βέλτιστη μέθοδος για την λύση τους μπορεί να βρεθεί μόνο μέσα από έρευνα και πειράματα.

Κεφάλαιο 2ο: Παραδείγματα εφαρμογών CSP

2.1 Εισαγωγή

Οι τεχνικές που συζητήθηκαν, στο προηγούμενο κεφάλαιο, για την επιτάχυνση της αναζήτησης μιας λύσης είναι αποτέλεσμα έρευνας πολλών χρόνων από πολλούς ερευνητές. Η ανάγκη για επιτάχυνση της αναζήτησης λύσεων των CSP πηγάζει από την πολυπλοκότητα που έχουν τα πραγματικά προβλήματα τα οποία λύνουν. Το γεγονός αυτό μαζί με την πολυπλοκότητα NP-Complete [1-3] της αναζήτησης κάνει τις τεχνικές αυτές τον μόνο εφικτό τρόπο να βρεθεί λύση σε αυτά τα προβλήματα.

2.2 Το Πρόβλημα του Χρονοπρογραμματισμού

Τα Προβλήματα Χρονοπρογραμματισμού (scheduling) είναι μια γενική κατηγορία προβλημάτων που συμπεριλαμβάνουν την ανάθεση γεγονότων (events) σε χρονοθυρίδες (timeslots) [1]. Τα προβλήματα αυτά έχουν περιορισμούς σχετικούς με τα γεγονότα, όπως "δύο γεγονότα δεν είναι συμβατά", "δύο γεγονότα πρέπει να συμβούν την ίδια στιγμή" ή "τα γεγονότα πρέπει να γίνουν με συγκεκριμένη σειρά". Οι περιορισμοί μπορεί να είναι ισχυροί ή ελαστικοί, άρα ο χρονοπρογραμματισμός είναι ένα πρόβλημα SCSP. Ο σκοπός των μοντέλων χρονοπρογραμματισμού είναι να ικανοποιήσουν όλους ισχυρούς περιορισμούς τηρώντας όσους περισσότερους ελαστικούς είναι δυνατόν.

Ένα παράδειγμα προβλήματος χρονοπρογραμματισμού είναι το Πρόβλημα Πανεπιστημιακού Προγράμματος Μαθημάτων (UCTTP - University Course Timetabling Problem) [1]. Το UCTTP είναι ένα πολύ σημαντικό πρόβλημα, καθώς τα Πανεπιστήμια πρέπει κάθε εξάμηνο να δημιουργούν ένα πρόγραμμα το οποίο πρέπει να οργανώνει μια πληθώρα από παράγοντες έτσι ώστε να τελεστεί σωστά η εκπαιδευτική διαδικασία. Οι παράγοντες αυτοί περιλαμβάνουν:

- **Γεγονότα**, όπως διαλέξεις, καθηγητές και φοιτητές.
- **Χρονοθυρίδες** στις οποίες μπορούν να συμβούν γεγονότα.
- **Πόρους**, πού απαιτούνται για τα μαθήματα όπως αίθουσες, εργαστήρια και εργαστηριακός εξοπλισμός.

Η οργάνωση του προγράμματος εξαρτάται από περιορισμούς που καθορίζουν πως τα γεγονότα οι χρονοθυρίδες και οι πόροι αλληλεπιδρούν μεταξύ τους. Οι περιορισμοί αυτοί είναι ισχυροί, όπως:

- Ένας καθηγητής/φοιτητής δεν μπορεί να είναι σε δύο αίθουσες/χρονοθυρίδες ταυτόχρονα.
- Δεν μπορούν να γίνονται δύο ή παραπάνω διαλέξεις στην ίδια αίθουσα την ίδια στιγμή.
- Σε κάθε αίθουσα χωράει συγκεκριμένος αριθμός από φοιτητές.
- Κάποιες διαλέξεις απαιτούν την χρήση ειδικού εξοπλισμού.
- Ο κάθε πόρος μπορεί να χρησιμοποιηθεί μια φορά σε μια χρονοθυρίδα.
- Και άλλοι.

Υπάρχουν και ελαστικοί περιορισμοί όπως:

- Οι καθηγητές μπορούν να ζητήσουν συγκεκριμένες ώρες και αίθουσες για τις διαλέξεις τους.
- Οι διαλέξεις πρέπει να γίνονται με τέτοια σειρά έτσι ώστε να ελαχιστοποιούνται οι κενές ώρες για τους καθηγητές και τους φοιτητές.
- Δεν πρέπει να υπάρχουν πολλές διαλέξεις τις βραδινές ώρες.
- Ο κάθε καθηγητής/φοιτητής δεν πρέπει να έχει πάνω από τέσσερις ώρες την μέρα στο πρόγραμμά του.

- Και άλλοι.

Η λύση του UCTTP είναι ένα εβδομαδιαίο ή μηνιαίο πρόγραμμα που αναθέτει γεγονότα και πόρους σε χρονοθυρίδες ικανοποιώντας κάθε ισχυρό περιορισμό και βελτιστοποιώντας όσους περισσότερους ελαστικούς περιορισμούς γίνεται. Δεν θεωρείται ότι υπάρχει ιεραρχία στους ελαστικούς περιορισμούς [1]. Έτσι η αντικειμενική συνάρτηση είναι το άθροισμα των παραβιασμένων ελαστικών περιορισμών και πρέπει να ελαχιστοποιηθεί. Το μοντέλο που θα μελετηθεί εξαρτάται από τις ανάγκες του εκάστοτε πανεπιστημίου. Η παραπάνω διατύπωση δίνει μια γενική ιδέα, αλλά οι μεταβλητές, οι περιορισμοί και οι στόχοι διαφέρουν από εργασία σε εργασία.

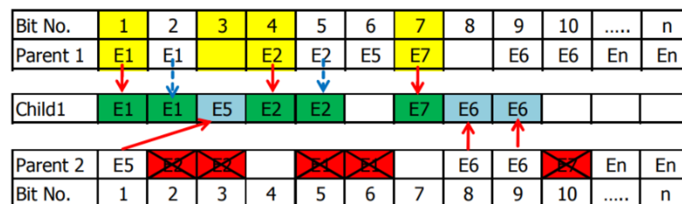
Το UCTTP αρχικά μοντελοποιήθηκε, σε μια απλή μορφή του, και λύθηκε ως πρόβλημα Χρωματισμού Γράφου [45]. Οι διαλέξεις συμβολίζονται από κόμβους που ενώνονται με ακμή αν δεν μπορούν να συμβούν την ίδια μέρα και οι μέρες είναι τα χρώματα. Η λύση δίνεται μέσω της κατασκευής υπογραφημάτων στα οποία κανένας κόμβος δεν είναι γειτονικός με άλλους, άρα το κάθε υπογράφημα μπορεί να πάρει το ίδιο χρώμα σε κάθε κόμβο του. Η μέθοδος βελτιώνεται για να προστεθούν και ελαστικοί περιορισμοί [46,47] και να δοθούν αποτελέσματα καλύτερης ποιότητας.

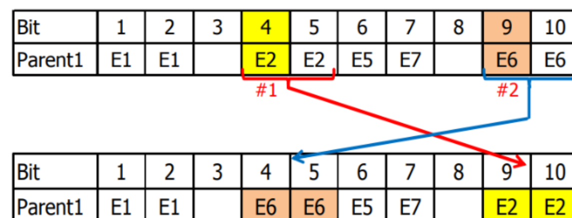
Το UCTTP λύνεται με CSP στην [49] χρησιμοποιώντας αντικειμενοστραφή προγραμματισμό για να υλοποιήσει τις τεχνικές CSP. Το μοντέλο περιέχει τα σύνολα (S, D, E, R), όπου S είναι το σύνολο των μαθημάτων (μεταβλητές), D είναι τα πεδία ορισμού των μεταβλητών στον χρόνο (χρονοθυρίδες), E είναι τα πεδία ορισμού των μεταβλητών στον χώρο (αίθουσες) και R οι περιορισμοί μεταξύ των μεταβλητών. Μοντελοποιούνται συγκεκριμένοι περιορισμοί και είναι όλοι ισχυροί. Επειδή υπάρχουν δύο διαστάσεις στις μεταβλητές χώρος και χρόνος, το πραγματικό πεδίο ορισμού τους είναι το καρτεσιανό γινόμενο των δύο πεδίων ορισμού. Οι πιθανές τιμές που πρέπει να ελεγχθούν για κάθε μεταβλητή είναι ζευγάρια χρονοθυρίδας και αίθουσας. Χρησιμοποιείται αλγόριθμος Backtracking με Forward Checking για να αποφεύγονται τα αδιέξοδα. Το FC παίζει σημαντικό ρόλο γιατί αν στην τρέχουσα κατάσταση έχει ήδη επιλεγεί μια αίθουσα δεν υπάρχει νόημα στο να ξαναεπιλεγεί. Στο πρόγραμμα χρησιμοποιούνται και τεχνικές διάταξης για τις μεταβλητές και τις τιμές τους. Οι μεταβλητές με την μεγαλύτερη συμμετοχή σε περιορισμούς (ή μεγαλύτερο βαθμό στο γράφημα περιορισμών) επιλέγονται πρώτες. Για τις τιμές, οι χρονοθυρίδες ταξινομούνται χρονικά, έτσι ώστε να ελαχιστοποιηθεί η χρήση τους και να μην μένουν κενά. Οι αίθουσες ταξινομούνται με κριτήριο αν η χωρητικότητα τους είναι κοντά στον αριθμό των φοιτητών που θα παρακολουθήσουν το μάθημα. Για τις τιμές μπορούν να δοθούν και προτιμήσεις ανά μάθημα, αυτές αλλάζουν την προτεραιότητα τους στην διάταξη. Οι μεταβλητές περιορισμοί και αλγόριθμος υλοποιήθηκαν σαν αντικείμενα στην γλώσσα C++.

Η [48] χρησιμοποιεί μεθόδους CSP για να λύσει το UCTTP, χρησιμοποιώντας το πρόγραμμα ILOG [59]. Το μοντέλο περιέχει χρονοθυρίδες και αίθουσες σαν μεταβλητές και αναθέτει διαλέξεις σε αυτές. Οι περιορισμοί μοντελοποιούνται με ιεραρχία και χωρίς, στα πειράματα ελέγχονται και οι δύο εκδοχές. Το πρόγραμμα χρησιμοποιεί την μέθοδο του FC με AC-5 για έλεγχό συνέπειας τόξου. Μελετούνται διάφορες οργανώσεις της μεθόδου διάταξης τιμών των μεταβλητών για να βρεθεί η καλύτερη. Τα πειράματα γίνονται με δείγματα από πραγματικά δεδομένα τμήματος πανεπιστημίου.

Στην [50] χρησιμοποιείται ένας υβριδικός αλγόριθμος που βασίζεται σε CSP και GA. Το μοντέλο είναι ίδιο με την [48]. Η διαδικασία που ακολουθείται είναι: ο GA παράγει κάποια λύση που είναι σχεδόν αποδεκτή, μετά η λύση αυτή επιδιορθώνεται με προώθηση περιορισμών και ελέγχεται για την ποιότητα της. Η διαδικασία επαναλαμβάνεται μέχρι να βρεθεί λύση ικανοποιητικής ποιότητας. Η επιδιόρθωση της λύσης γίνεται επιλέγοντας μια τυχαία τιμή για κάθε μεταβλητή που παραβιάζει έναν περιορισμό μέχρι η νέα τυχαία τιμή να είναι αποδεκτή. Μόλις βρεθεί η νέα τιμή αυτή αφαιρείται από τα πεδία των άλλων μεταβλητών με FC.

Μια εργασία που χρησιμοποιεί μόνο GA είναι η [51]. Χρησιμοποιεί πληθυσμό δύο ατόμων μόνο και οι μέθοδοι για τον συνδυασμό και μετάλλαξη είναι λίγο διαφορετικοί από αυτές που περιγράφηκαν στο προηγούμενο κεφάλαιο, θα συμβεί είτε συνδυασμός είτε μετάλλαξη. Ο συνδυασμός των δύο ατόμων φτιάχνει δύο παιδιά, για κάθε παιδί αντιγράφονται τυχαίες θέσεις από έναν γονέα σε ένα κενό παιδί, οι υπόλοιπες κενές συμπληρώνονται με τιμές από τυχαίες θέσεις του άλλου γονέα. Η μετάλλαξη θα κάνει ανταλλαγή ενός ζευγαριού θέσεων, του κάθε ατόμου, μεταξύ τους, τα μεταλλαγμένα άτομα εισάγονται στον πληθυσμό σαν "παιδιά". Έτσι ο πληθυσμός έχει δύο γονείς και δύο παιδιά, από αυτούς επιλέγονται οι δύο καλύτεροι βάσει της συνάρτησης καταλληλότητας. Η συνάρτηση καταλληλότητας βασίζεται στην παραβίαση ισχυρών και ελαστικών περιορισμών. Ο αλγόριθμος δεν ελέγχει αν η τελική λύση παραβιάζει ισχυρούς περιορισμούς, αλλά στην πράξη στα αποτελέσματα αυτό δεν συμβαίνει γιατί μπαίνει βάρος 10^4 στο ολικό κόστος των ισχυρών περιορισμών και 1 στο κόστος των ελαστικών περιορισμών. Τα πειράματα δείχνουν ότι μια καλή πιθανότητα συνδυασμού, αντί για μετάλλαξη, είναι 0.7.



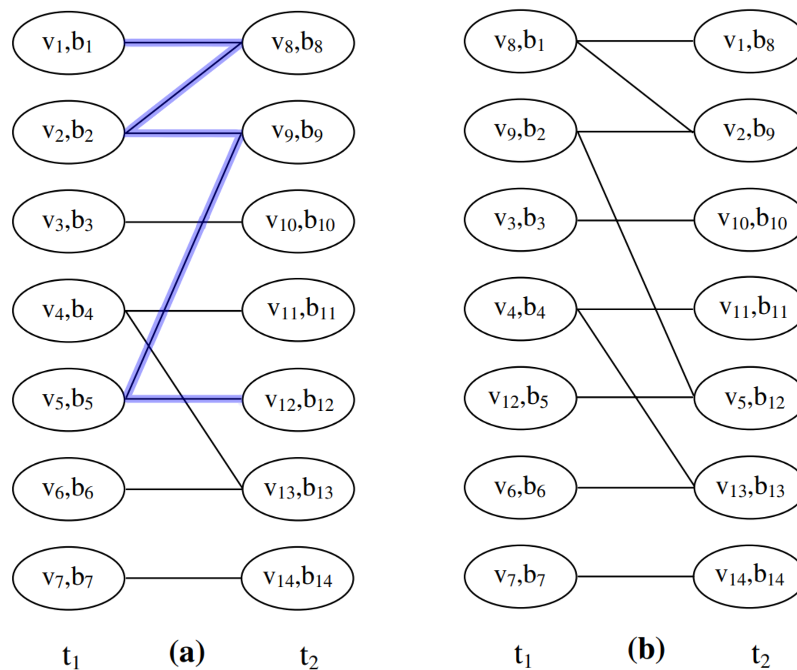


Σχήμα 22.12: Διαδικασίες συνδυασμού και μετάλλαξης στην εργασία [51].

Μια ακόμα εφαρμογή του GA για το UCTTP βρίσκεται στην [52]. Εδώ τα χρωμοσώματα κάθε ατόμου αποτελούνται από πεντάδες (S_i, S_{ei}, D_i, T_i, R_i) όπου S_i είναι η επιλογή μαθήματος, S_{ei} είναι η

επιλογή διάλεξης, D_i είναι ο καθηγητής, T_i ο χρόνος και R_i η αίθουσα. Μια πεντάδα είναι μια εγγραφή στο τελικό πρόγραμμα. Ο πληθυσμός ξεκινά με άτομα που δεν παραβιάζουν κανέναν ισχυρό περιορισμό και προσπαθεί να βελτιώσει τους ελαστικούς περιορισμούς που παραβιάζονται. Ο συνδυασμός ατόμων γίνεται επιλέγονται τυχαία κάποιους από τους καλύτερους και ακολουθώντας την διαδικασία που παρουσιάστηκε στο σχήμα. Όπως και στην [51] ο αλγόριθμος δεν απαγορεύει λύσεις που παραβιάζουν κάποιον ισχυρό περιορισμό, αλλά αποτρέπει την εμφάνιση τους δίνοντας τους μεγαλύτερο βάρος στην συνάρτηση καταλληλότητας.

Την μέθοδο της SA χρησιμοποιεί η [57], χρησιμοποιώντας ελαστικό περιορισμό στην θέση του ισχυρού περιορισμού αριθμού χρονοθυρίδων και ψάχνοντας για μια λύση μηδενικού κόστους. Μια λύση μηδενικού κόστους με ελαστικό περιορισμό αντιστοιχεί σε μία αποδεκτή λύση ισχυρού περιορισμού. Εδώ επιτρέπεται ο αλγόριθμος να βάλει γεγονότα σε πλασματικές χρονοθυρίδες και το κόστος είναι ο αριθμός τους. Η αναπαράσταση του μοντέλου περιλαμβάνει γεγονότα v_i και αναθέτει χρονοθυρίδες t_i και πόρους b_i σε αυτά. Ο αλγόριθμος κάνει κινήσεις στον χώρο πλήρων αναθέσεων, σε τρία είδη γειτονιών: απλή γειτονία που απλά αλλάζει κάποιον πόρο ενός γεγονότος σε κάποιον άλλο αποδεκτό, γειτονιά ανταλλαγής που ανταλλάσσει τους πόρους δύο γεγονότων και γειτονιά αλυσίδας Kempe. Το τελευταίο είδος γειτονιάς χρησιμοποιείται για να ξεφύγει από τα τοπικά ελάχιστα, κάνει ανταλλαγές σε μία αλυσίδα από γεγονότα όπως φαίνεται στο σχήμα. Η θερμοκρασία (πιθανότητα αποδοχής κίνησης που αυξάνει το κόστος) είναι αρχικά 40% και σε κάθε τοπικό ελάχιστο ψύχεται με την συνάρτηση $T = 1/((1/T\ ) + \beta)$ όπου β είναι σταθερά επιλέγεται μεταξύ 0.001 και 0.0005.



Σχήμα 22.13: Παράδειγμα αλυσίδας Kempe από την [57].

Στο γράφημα περιορισμών (a) για τις μεταβλητές v_1 μέχρι v_{14} οι κόμβοι ενώνονται μόνο αν υπάρχει σύγκρουση μεταξύ τους. Στην κάθε μεταβλητή έχει ανατεθεί ένας πόρος b_i στην τρέχουσα κατάσταση, οι μεταβλητές είναι χωρισμένες σε χρονοθυρίδες t_1 και t_2. Η διαδικασία αλυσίδας Kempe που ακολουθείται στην [57] επιλέγει

τυχαία έναν κόμβων στην t_1 (έστω τον v_5) και παίρνει το υπογράφημα όσων συνδέονται με αυτόν άμεσα και έμμεσα (φαίνεται με μπλε ακμές στο σχήμα). Αυτή είναι η αλυσίδα Kempe, για να κάνει βήμα σε νέα κατάσταση ο αλγόριθμος ανταλλάσσει τους πόρους b_i στα ζευγάρια που ενώνονται με οριζόντια ακμή (v_1 και v_8, v_2 και v_9, v_5 και v_{12}) (b).

Εφαρμογή του TS κάνει η [58], οι μεταβλητές είναι ένας τρισδιάστατος πίνακας όπου το στοιχείο x_{ita} είναι 1 αν το μάθημα i γίνεται στην αίθουσα a στην χρονοθυρίδα t. Δίνεται μια αντικειμενική συνάρτηση με βάρη για κάθε περιορισμό που παραβιάζεται, όλοι οι περιορισμοί είναι ισχυροί αλλά οργανωμένοι σε ιεραρχία. Η λίστα ταμπού θυμάται τις τελευταίες K κινήσεις μαθημάτων μεταξύ αιθουσών και χρονοθυρίδων που έχουν γίνει και απαγορεύει την αντίθετη κίνηση, λειτουργεί με ουρά FIFO. Το κριτήριο φιλοδοξίας επιτρέπει κινήσεις ταμπού μόνο αν βελτιώνουν την αντικειμενική συνάρτηση. Οι κινήσεις γίνονται στην απλή γειτονιά (το μάθημα μεταφέρεται σε άλλη χρονοθυρίδα) ή την γειτονιά ανταλλαγής (δύο μαθήματα ανταλλάσσουν χρονοθυρίδα). Περιορισμένος αριθμός από γείτονες εξετάζονται γιατί οι γειτονιές είναι μεγάλες και επιλέγεται ο καλύτερος βάσει της αντικειμενικής συνάρτησης. Η τελευταία καθολικά καλύτερη λύση αποθηκεύεται, αν δεν βελτιωθεί για N βήματα ο αλγόριθμος ξαναξεκινά από αυτήν. Αν κολλήσει σε τοπικό ελάχιστο, ο αλγόριθμος ξαναρχίζει από μια τυχαία κατάσταση. Η αναζήτηση σταματά όταν η αντικειμενική αξία μιας λύσης γίνει μηδέν. Διαφορές οργανώσεις για επιλογή επόμενου βήματος μελετούνται σε πειράματα.

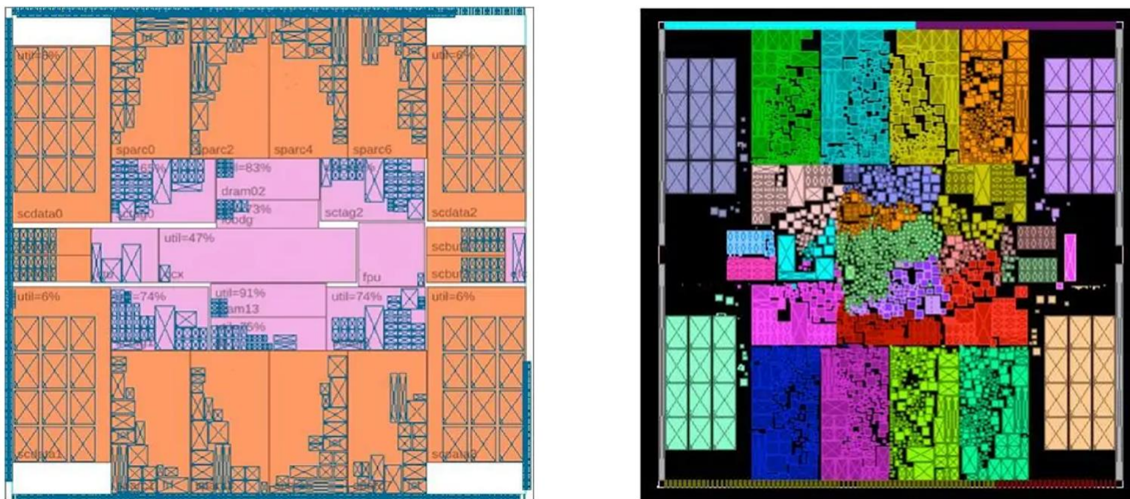
Το πρόβλημα του χρονοπρογραμματισμού δεν έχει εφαρμογές μόνο στην δημιουργία προγράμματος μαθημάτων. Είναι προφανές ότι απλά αλλάζοντας τις έννοιες των γεγονότων και πόρων μπορούν να φτιαχτούν προγράμματα για οποιοδήποτε πρόβλημα χρειάζεται κάποια οργάνωση στον χρόνο. Οι μέθοδοι και τεχνικές που συζητήθηκαν παραπάνω μπορούν να εφαρμοστούν και σε άλλα σενάρια, με αλλαγές στους περιορισμούς.

Η εργασία [8] πραγματοποιεί τον χρονοπρογραμματισμό των δρομολογίων ενός σιδηρόδρομου για την χρήση τους από τρένα. Τα γεγονότα, ή μεταβλητές, σε αυτό το μοντέλο είναι η κίνηση των τρένων πάνω από ένα κομμάτι σιδηρόδρομου και πόροι είναι τα κομμάτια σιδηρόδρομου. Για κάθε τρένο που πρέπει να δρομολογηθεί δημιουργείται μια ακολουθία από μεταβλητές t_{ci} που μόλις τους ανατεθούν κομμάτια σιδηρόδρομου η ακολουθία δείχνει την πλήρη διαδρομή αυτού του τραίνου. Δημιουργούνται αρκετές έτσι ώστε να μπορεί να καλυφθεί η μέγιστη πιθανή διαδρομή, αν η τελική διαδρομή είναι πιο σύντομη, κάποιες συμπληρώνονται με ένα νοητό κομμάτι. Ακόμα δημιουργούνται οι μεταβλητές έναρξης και λήξης του ταξιδιού, ώστε να μπορεί να υπολογιστεί ποια κομμάτια του σιδηρόδρομου χρησιμοποιούνται κάθε στιγμή. Ο αλγόριθμος χρησιμοποιεί B&B με Backtracking για να ελαχιστοποιήσει τα μεγέθη των διαδρομών και κάνει προώθηση περιορισμών με FC. Οι μεταβλητές συμπληρώνονται με την σειρά: tci → χρόνος έναρξη → χρόνος λήξη. Οι τιμές των tci εξετάζονται στατικά, οι τιμές των μεταβλητών χρόνου εξετάζονται με αύξουσα σειρά. Έγιναν πειράματα με το μοντέλο χρησιμοποιώντας πραγματικά δεδομένα, το μοντέλο σε μερικά λεπτά καταφέρνει να βρει λύση με παραπάνω από 60% λιγότερη καθυστέρηση από τα δεδομένα.

2.3 Σχεδίαση Υλικού

Ο τομέας VLSI (Very Large Scale Integration) ασχολείται με την σχεδίαση υλικού υπολογιστών (hardware). Το υλικό αυτό απαιτείται να έχει μεγάλη πυκνότητα σε transistor για να έχει την απαραίτητη επεξεργαστική ισχύ [38]. Λόγω του εξαιρετικά μεγάλου αριθμού από transistor και καλωδίων που πρέπει να τοποθετηθούν σε ένα κύκλωμα η σχεδίαση του από έναν άνθρωπο είναι αδύνατη. Ήδη από την δεκαετία του ‘90 είχε ξεκινήσει έρευνα για την αυτοματοποίηση τις σχεδίασης υλικού φυσικού επιπέδου (floorplanning) βασισμένο σε σχέδιο υψηλότερου επιπέδου από έναν σχεδιαστή [39].

Το πρόβλημα του floorplanning είναι ένα πρόβλημα με περιορισμούς καθώς πρέπει τα στοιχεία του κυκλώματος να τοποθετηθούν στο χώρο χωρίς να υπάρχει επικάλυψη μεταξύ τους. Τα στοιχεία αυτά είναι συνήθως είδη σχεδιασμένα κυκλώματα, όπως αθροιστές, μνήμες, μεμονωμένες λογικές πύλες και άλλα. Η δημιουργία του floorplan γίνεται από ένα πρόγραμμα σχεδίασης αυτόματα αφού ο σχεδιαστής δώσει το σχέδιο σε μορφή RTL (Register Transfer Level) και τους περιορισμούς [43]. Ακόμη είναι σημαντικό οι αγωγοί που συνδέουν τα μέρη του κυκλώματος να έχουν το ελάχιστο μήκος καθώς αυτό κάνει το κύκλωμα πιο γρήγορο και η μικρή συνολική έκταση μειώνει το κόστος κατασκευής [40]. Τα προβλήματα του floorplanning είναι NP-Complete και μπορούν να χρησιμοποιηθούν τεχνικές CSOP για να λυθούν.



Σχήμα 22.14: Παραδείγματα από αποτελέσματα εφαρμογών floorplanning [43].

Οι τεχνικές CSOP που χρησιμοποιούνται για να λυθεί το πρόβλημα του floorplanning είναι οι SA [22,41] και GA [42]. Και οι δύο προσεγγίσεις βασίζονται σε μια αναπαράσταση του χώρου βάσει της οποίας μπορεί να διαιρεθεί η επιφάνεια του κυκλώματος σε περιοχές και μετά να τοποθετηθούν τα στοιχεία σε αυτές. Η αναπαράσταση αυτή είναι η μέθοδος με την οποία τηρούνται οι περιορισμοί του προβλήματος, αυτή παράγει εφικτές λύσεις και οι αλγόριθμοι ψάχνουν τις βέλτιστες. Καθώς το

floorplanning θα συνεχίσει να είναι πολύ σημαντική αυτοματοποίηση της σχεδίασης υπολογιστών στο μέλλον, η έρευνα για την ανάπτυξη καλύτερων αλγορίθμων βελτιστοποίησης του συνεχίζεται [40].

2.4 Γλώσσες Μοντελοποίησης

Οι τεχνικές που συζητήθηκαν δεν θα ήταν χρήσιμες αν δεν γινόταν να υλοποιηθούν σε ένα πρόγραμμα το οποίο θα εκτελέσει ένας υπολογιστής. Όπως με όλους τους αλγορίθμους ένα πρόγραμμα επίλυσης CSP γίνεται να γραφεί σε μια οποιαδήποτε γενική γλώσσα προγραμματισμού όπως C, C++, Java ή Python. Για να επιταχυνθεί όμως η δημιουργία πρωτοτύπων για πειράματα, να μειωθεί το επίπεδο δυσκολίας υλοποίησης εφαρμογών και να βελτιωθεί η απόδοση των τελικών προγραμμάτων, έχει γίνει έρευνα για την δημιουργία γλωσσών και βιβλιοθηκών προγραμματισμού περιορισμών.

Μια πλήρης εικόνα της επίλυσης CSP με Haskell δίνεται στην [62]. Παρουσιάζεται πως μπορούν να χρησιμοποιηθούν τα monads της Haskell για να υλοποιηθούν ισχυρές δομές επίλυσης του μοντέλου και παραδείγματα για την υλοποίηση τεχνικών αναζήτησης. Το σύστημα επίλυσης που υλοποιείται είναι αρκετά ευέλικτο έτσι ώστε να μπορεί να προσαρμοστεί σε μια πληθώρα από μοντέλα.

Οι ILOG [59] και Gecode [60] είναι βιβλιοθήκες για την C++ που επιτρέπουν την δημιουργία προγραμμάτων επίλυσης CSP. Οι βιβλιοθήκες δίνουν στον χρήστη την δυνατότητα να υλοποιήσει προγράμματα CSP χωρίς να χρειάζεται να μάθει μια καινούργια γλώσσα. Το μοντέλο και το πρόγραμμα επίλυσης (solver) εκφράζονται μέσα από αντικείμενα. Αυτό επιτρέπει στον χρήστη να εκφράσει το μοντέλο και χρησιμοποιήσει τις τεχνικές επίλυσης ασχολούμενος μόνο με την επιλογή μεθόδου και τις παραμέτρους της. Η χρήση βιβλιοθηκών έχει και αρνητικά, προγραμματίζοντας σε μια γλώσσα ο χρήστης πρέπει να ακολουθεί τα μοτίβα και τις συμβάσεις της [63]. Ακόμη οι γενικές γλώσσες δεν είναι σχεδιασμένες γύρω από τις έννοιες της αναζήτησης και των περιορισμών, άρα ο χρήστης πρέπει να χειρίζεται όλες τις λεπτομέρειες για να μπορέσει να εκμεταλλευτεί τις βιβλιοθήκες.

Ένα σημαντικό θετικό χαρακτηριστικό των μοντέρνων γλωσσών προγραμματισμού είναι ότι υποστηρίζουν παράλληλη επεξεργασία. Αυτό κάνει εύκολη την υλοποίηση παράλληλων προγραμμάτων για την επίλυση CSP [64]. Η αναζήτηση είναι μια διαδικασία που μπορεί να παραλληλοποιήθει εύκολα: αντί για ψάχνει σε ένα μέρος στο δέντρο ο αλγόριθμος μπορεί να χωρίσει το δέντρο σε υπό-δέντρα και να ψάξει σε όλα ταυτόχρονα. Όσο για την τοπική αναζήτηση ήδη είναι σχεδιασμένη έτσι ώστε να ψάχνει σε διάφορους τόπους του χώρου αναζήτησης (τυχαία επανεκκίνηση) άρα αναζητώντας ταυτόχρονα σε πολλά μέρη κάνει την διαδικασία πιο γρήγορη. Η εργασία [65] προτείνει έναν συνδυασμό διαφόρων εργαλείων για την παραλληλοποίηση της αναζήτησης: Bobpp [64], Gecode [60] και OR-Tools [61] (το οποίο θα παρουσιαστεί και θα χρησιμοποιηθεί σε παρακάτω κεφάλαιο).

Η Turtle [63] είναι μια γλώσσα σχεδιασμένη για την συγγραφή προγραμμάτων επίλυσης CSP. Η Trutle επιτρέπει στον χρήστη να εκφράζει τα μοντέλα και τους περιορισμούς του ελεύθερος από τις αναγκαστικές λεπτομέρειες των κοινών γλωσσών. Παρακάτω φαίνεται ένα παράδειγμα της Turtle (b) σε σύγκριση με μια κοινή γλώσσα (a):

```
(a)
1     while (mouse.pressed)
2     { //message processing is left out
3         int y = mouse.y;
4         if (y > border.max)
5             y = border.max;
6         if (y < border.min)
7             y = border.min;
8         draw_element (fix_x,y,graphic);
9     }
(b)
1     while (mouse.pressed) {
2         constrained <int> y = mouse.y;
3         require ( y >= border.min &&
4                      y <= border.max);
5         draw_element (fix_x,y(),graphic);
6     }
```

Φαίνεται η εκφραστική δύναμη της Trutle καθώς ο χρήστης δεν χρειάζεται να υλοποιήσει μια διαδικασία για να επιβάλλει τους περιορισμούς. Αντί γι' αυτό εκφράζει τους περιορισμούς και αφήνει την γλώσσα να χειριστεί τις λεπτομέρειες.

Η OPL είναι ένα ακόμα παράδειγμα γλώσσας για επίλυση CSP [6]. Περιέχει αλγεβρικές συναρτήσεις και συναρτήσεις συνόλων υψηλού επιπέδου και επιτρέπει στον χρήστη να υλοποιήσει τις λεπτομέρειες του μοντέλου με εκφράσεις υψηλού επιπέδου. Η εργασία παρουσιάζει εφαρμογές της OPL στην επεξεργασία σήματος και στον χρονοπρογραμματισμό.

Το Oz Explorer [66] είναι ένα διαδραστικό πρόγραμμα επίλυσης CSP. Στην διεπαφή του το πρόγραμμα οπτικοποιεί το δέντρο αναζήτησης για τον χρήστη. Αυτό δίνει στον χρήστη μια πιο απτή εικόνα του προβλήματος που έχει να λύσει, βοηθώντας τον να καταλάβει ανεπάρκειες ή λάθη στο πρόγραμμα του και να το βελτιώσει.

Σχήμα 22.15: Παράδειγμα του παραθύρου του Oz Explorer [66].

2.5 Παιχνίδια – Παζλ

Οι μέθοδοι επίλυσης CSP μπορούν να εφαρμοστούν σε λογικούς γρίφους ή παζλ. Τα παιχνίδια αυτά παίζονται από έναν παίκτη, ο οποίος προσπαθεί να βρει μια λύση ακολουθώντας τους κανόνες. Μπορεί να παίρνουν μορφή τυπωμένου γρίφου (όπως το sudoku [53]), φυσικού αντικειμένου (όπως ο κύβος του Ρούμπικ [54]) ή ψηφιακή (όπως ο Ναρκαλιευτής [55]). Η μορφή του παζλ μπορεί να μετατραπεί σε ένα μοντέλο, με τους κανόνες σαν περιορισμούς, και να λυθεί αυτόματα από έναν υπολογιστή. Συνήθως η αυτόματη λύση παιχνιδιών δεν αποτελεί θέμα ακαδημαϊκής έρευνας επειδή δεν έχουν κάποια πρακτική εφαρμογή. Υπάρχουν εξαιρέσεις όπως η [54] που αποδεικνύει ότι ένας κύβος του Ρούμπικ μπορεί να λυθεί το πολύ σε 20 κινήσεις και ταυτόχρονα επιδεικνύει την δύναμη των μοντέρνων υπολογιστών και αλγορίθμων αναζήτησης. Παρά την έλλειψη πρακτικών εφαρμογών η μοντελοποίηση τους είναι καλή εξάσκηση των αρχών του CSP. Στο επόμενο κεφάλαιο δίνεται σαν παράδειγμα η λύση του παιχνιδιού “Skyscrapers”.

2.6 Συμπεράσματα

Σε αυτό το κεφάλαιο δόθηκε μια επισκόπηση των προβλημάτων που λύνονται με την χρήση τεχνικών CSP. Το πρόβλημα του χρονοπρογραμματισμού είναι πολύ σημαντικό καθώς περιέχει πολλούς πιθανούς συνδυασμούς, κάνοντας την εύρεση μιας λύσης δύσκολη. Η προσθήκη ελαστικών περιορισμών περιπλέκει το πρόβλημα ακόμα περισσότερο. Απαιτείται η επιλογή κάποιων περιορισμών για παραβίαση σε αναζήτηση μιας λύσης καλύτερης ποιότητας. Το πρόβλημα όμως έχει εφαρμογή σε πολλές πτυχές της καθημερινότητας, καθώς η οργάνωση των γεγονότων σε ένα πρόγραμμα είναι κρίσιμο κομμάτι της λειτουργίας σχολείων, πανεπιστημίων, επιχειρήσεων και της βιομηχανίας. Τέλος παρουσιάζονται οι γλώσσες προγραμματισμού που χρησιμοποιούνται για την ανάπτυξη εφαρμογών CSP και φαίνονται τα θετικά και αρνητικά των δύο ειδών: γενικές και ειδικές γλώσσες.

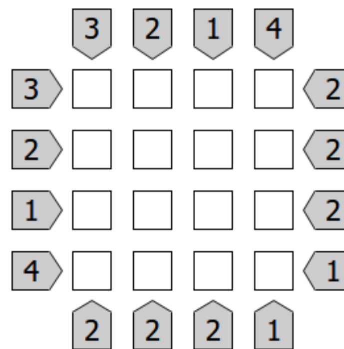
Κεφάλαιο 3ο: Μοντελοποίηση του παιχνιδιού “Skyscrapers”

3.1 Εισαγωγή

Το πρακτικό κομμάτι αυτής της εργασίας περιλαμβάνει την μοντελοποίηση και στην συνέχεια επίλυση του παιχνιδιού “Skyscrapers” με χρήση προγραμματισμού ικανοποίησης περιορισμών. Σε αυτό το κεφάλαιο παρουσιάζονται οι κανόνες του Skyscrapers, η μοντελοποίηση τους σε ένα μαθηματικό μοντέλο, η ανάπτυξη ενός προγραμματιστικού μοντέλου και τέλος η υλοποίηση του προγραμματιστικού μοντέλου σε ένα πρόγραμμα που λύνει το παιχνίδι.

3.2 Κανόνες του “Skyscrapers”

Το “Skyscrapers” γνωστό και ως “Towers” είναι ένα παιχνίδι-γρίφος λογικής. Το παιχνίδι αποτελείται από ένα τετραγωνικό πλέγμα διαστάσεων NxN με αριθμούς στην κάθε μία από τις τέσσερις πλευρές του. Ένα παράδειγμα του παιχνιδιού φαίνεται στο σχήμα 3.1.



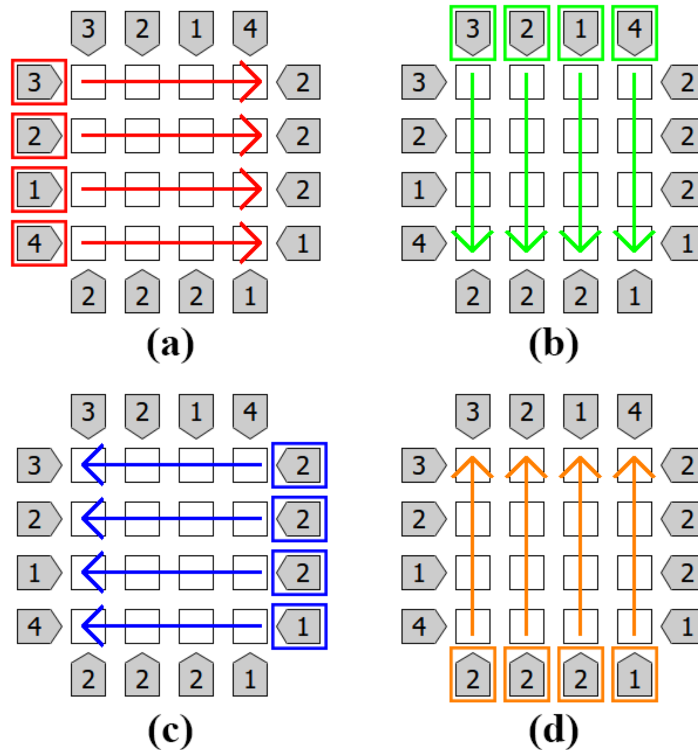
Σχήμα 33.16: Ασυμπλήρωτο πλέγμα Skyscrapers 4x4.

Το πλέγμα περιέχει N^2 κενά “οικοδομικά τετράγωνα” όπου ο παίκτης πρέπει να χτίσει ουρανοξύστες. Συμβολικά το κάθε τετράγωνο του πλέγματος συμπληρώνεται με έναν αριθμό ο οποίος συμβολίζει το ύψος του ουρανοξύστη που χτίστηκε εκεί. Οι αριθμοί στις τέσσερις πλευρές του πλέγματος αποτελούν στοιχεία που υποδεικνύουν στον παίκτη πως πρέπει να τοποθετηθούν οι ουρανοξύστες. Ο τελικός σκοπός του παίκτη είναι να γεμίσει το πλέγμα με ουρανοξύστες, ακολουθώντας τους παρακάτω κανόνες:

- Κανόνας-1: Κάθε ουρανοξύστης πρέπει να έχει ύψος στο εύρος από 1 έως N. Το ύψος είναι φυσικός αριθμός.
- Κανόνας-2: Δεν επιτρέπεται να τοποθετηθούν δύο ουρανοξύστες με το ίδιο ακριβώς ύψος στην ίδια οριζόντια γραμμή του πλέγματος.
- Κανόνας-3: Δεν επιτρέπεται να τοποθετηθούν δύο ουρανοξύστες με το ίδιο ακριβώς ύψος στην ίδια κάθετη στήλη του πλέγματος.
- Κανόνας-4: Ο παίκτης πρέπει να ικανοποιήσει το κάθε ένα από τους περιορισμούς στις πλευρές του πλέγματος. Δηλαδή πρέπει για το κάθε στοιχείο οι ουρανοξύστες που είναι “ορατοί” στην “κατεύθυνση” του να είναι ίσοι με τον αριθμό του.

- Κανόνας-5: Αν κάποιοι ουρανοξύστες έχουν ήδη χτιστεί (το πλέγμα περιέχει εξαρχής ουρανοξύστες τοποθετημένους σε κάποια τετράγωνα) τότε ο παίκτης δεν μπορεί να αλλάξει αυτούς τους ουρανοξύστες στην λύση του.

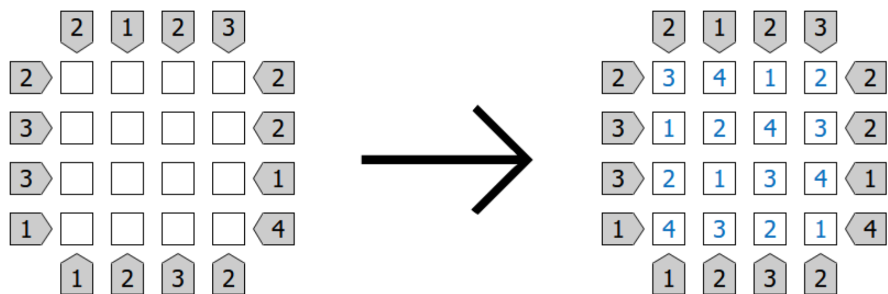
Όσων αφορά τον Κανόνα-4, για ένα στοιχείο "κατεύθυνση" σημαίνει η γραμμή/στήλη που του αντιστοιχεί, κοιτώντας από την πλευρά του προς την απέναντι πλευρά. Για παράδειγμα το στοιχείο της πρώτης γραμμής στη αριστερή πλευρά έχει κατεύθυνση την πρώτη γραμμή και βλέπει τους ουρανοξύστες από τα αριστερά προς τα δεξιά. Το στοιχείο της κάτω πλευράς και τέταρτης στήλης βλέπει τους ουρανοξύστες της τέταρτης στήλης από κάτω προς τα πάνω. Το σχήμα 3.2 δείχνει όλες τις κατευθύνσεις των στοιχείων του 4x4 πλέγματος του προηγούμενου παραδείγματος.



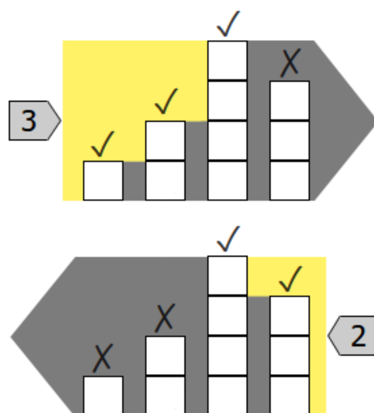
Σχήμα 33.17: Οι πλευρές "αριστερά" (a), "πάνω" (b), "κάτω" (c) και "δεξιά" (d).

Για ευκολία οι κατευθύνσεις θα αναφέρονται με το αντίστοιχο σημείο του ορίζοντα, έτσι τα στοιχεία της αριστερής πλευράς κοιτούν ανατολικά, της πάνω νότια, της δεξιά δυτικά και τις κάτω βόρεια.

Για να είναι "ορατός" ένας ουρανοξύστης σε μία κατεύθυνση του πρέπει να είναι είτε ο πρώτος της κατεύθυνσης ή να μην "κρύβεται" από κανέναν προηγούμενο του. Ένας ουρανοξύστης που είναι ψηλότερος από κάποιον επόμενο του θα τον "κρύβει". Ένα παράδειγμα της ορατότητας φαίνεται στα σχήματα 3.3 και 3.4.

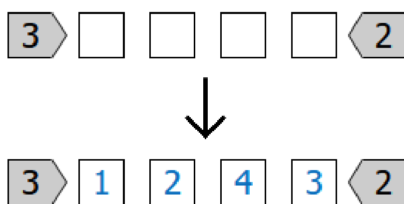


Σχήμα 33.18: Παράδειγμα λύσης ενός γρίφου 4x4.



Σχήμα 33.19: Ορατότητα από αριστερά και από δεξιά.

Σε αυτή την γραμμή από τέσσερις ουρανοξύστες, από αριστερά φαίνονται τρεις ουρανοξύστες καθώς ο τρίτος κρύβει τον τέταρτο ενώ από δεξιά είναι ορατοί δύο επειδή ο τρίτος (πάντα αριθμούμε από αριστερά προς τα δεξιά) κρύβει τον δεύτερο και τον πρώτο.



Σχήμα 33.20: Συμπλήρωση μιας γραμμής.

Αυτός είναι ένας από τους αποδεκτούς τρόπους να συμπληρωθεί η πρώτη γραμμή του παραδείγματος του παιχνιδιού που δόθηκε στο σχήμα 3.1.

Σημειώσεις για τους κανόνες:

- Οι κανόνες δεν εξασφαλίζουν ότι κάθε διάταξη των στοιχείων στις πλευρές του πλέγματος έχει μοναδική λύση, υπάρχουν διατάξεις με πολλαπλές λύσεις.
- Μια πιο δύσκολη εκδοχή του παιχνιδιού μπορεί να μην δίνει στον παίκτη όλα τα στοιχεία στις πλευρές του πλέγματος, αλλά να κρατάει κάποια από αυτά κρυφά. Συμβολίζουμε τα κρυφά στοιχεία με “0” και αγνοούμε τον Κανόνα-4 για αυτά.

3.3 Μαθηματική μοντελοποίηση

Για να οριστεί το "Skyscrapers" σαν Πρόβλημα Ικανοποίησης Περιορισμών πρέπει να οριστούν οι μεταβλητές και οι περιορισμοί του μοντέλου.

Αρχίζοντας με τις μεταβλητές, αυτές θα οριστούν σαν στοιχεία ενός NxN πίνακα V:

$$V_{mat} = \begin{bmatrix} v_{1,1} & v_{1,2} & v_{1,3} & \cdots & v_{1,N} \\ v_{2,1} & v_{2,2} & v_{2,3} & \cdots & v_{2,N} \\ v_{3,1} & v_{3,2} & v_{3,3} & \cdots & v_{3,N} \\ \cdots & \cdots & \cdots & & \cdots \\ v_{N,1} & v_{N,2} & v_{N,3} & \cdots & v_{N,N} \end{bmatrix}$$

(3.1)

Ο λόγος που γίνεται αυτό είναι διότι με αυτή την οργάνωση αντιστοιχίζονται ξεκάθαρα οι μεταβλητές του μοντέλου με τα τετράγωνα του πλέγματος του παιχνιδιού. Έτσι το τετράγωνο στην γραμμή i και στήλη j του πλέγματος αντιστοιχεί στην μεταβλητή $v_{i,j}$. Πέρα από την απλή οργάνωση των μεταβλητών σε γραμμές και στήλες δεν θα χρησιμοποιηθούν πράξεις γραμμικής άλγεβρας σε αυτό το μοντέλο. Το σύνολο που περιέχει όλες τις μεταβλητές είναι το V:

$$V = \{v_{i,j} \mid v_{i,j} \,\epsilon\, V_{mat} \;\forall i, j \epsilon [1, N]\}$$

(3.2)

Για κάθε μεταβλητή πρέπει να οριστεί και ένα πεδίο ορισμού, σύμφωνα με τον Κανόνα-1 οι ουρανοξύστες, άρα οι τιμές που συμπληρώνονται στα τετράγωνα, παίρνουν τιμές φυσικούς αριθμούς στο εύρος από 1 έως N. Άρα για την μεταβλητή $v_{i,j}$ το πεδίο ορισμού της είναι οι τιμές στο σύνολο $D_{i,j}$:

$$D_{i,j} = \left\{x \mid x \epsilon \mathbb{N}^+, \; x \epsilon [1, N]\right\}$$

(3.3)

Ο σκοπός του μοντέλου είναι να αναθέσει σε κάθε μεταβλητή του πίνακα V (σχέση 3.1) μια τιμή από το πεδίο ορισμού της. Για να γίνει αυτό βάσει των κανόνων πρέπει να οριστούν μερικοί περιορισμοί.

Ο πρώτος περιορισμός απαγορεύει σε δύο μεταβλητές στην ίδια γραμμή να έχουν την ίδια τιμή, όπως επιβάλλει ο Κανόνας-2, δίνεται στην σχέση 3.4:

$$C_1 : v_{a,j} \neq v_{b,j} \;\; \forall a, b \epsilon [1, N], \; a \neq b$$

(3.4)

Παρομοίως ο δεύτερος περιορισμός απαγορεύει σε δύο μεταβλητές στην ίδια στήλη να πάρουν την ίδια τιμή, για να τηρηθεί ο Κανόνας-3, δίνεται στην σχέση 3.5:

$$C_2 : v_{i,a} \neq v_{i,b} \;\; \forall a, b \epsilon [1, N], \; a \neq b$$

(3.5)

Σύμφωνα με τον Κανόνα-5 μπορεί να υπάρχουν κάποιες τιμές για συγκεκριμένα τετράγωνα που δίνονται εξ αρχής στον γρίφο, θεωρείται ότι οι τιμές αυτές δίνονται στον πίνακα P (σχέση 3.6) και ότι είναι έγκυρες ως προς τους άλλους κανόνες:

$$P_{mat} = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & \cdots & p_{1,N} \\ p_{2,1} & p_{2,2} & p_{2,3} & \cdots & p_{2,N} \\ p_{3,1} & p_{3,2} & p_{3,3} & \cdots & p_{3,N} \\ \cdots & \cdots & \cdots & & \cdots \\ p_{N,1} & p_{N,2} & p_{N,3} & \cdots & p_{N,N} \end{bmatrix}$$

$$p_{i,j} \epsilon [0, N] \;\; \forall i, j \epsilon [1, N]$$

(3.6)

Όπου αν η τιμή $p_{i,j}$ είναι μεγαλύτερη του μηδενός ανήκει στο σύνολο P^+ (σχέση 3.7) και πρέπει να ανατεθεί αναγκαστικά στην μεταβλητή $v_{i,j}$, οπότε ορίζουμε τον τρίτο περιορισμό:

$$P^+ = \{p_{a,b} \mid p_{a,b} > 0 \;\; \forall a, b \epsilon [1, N]\}$$

(3.7)

$$C_3 : v_{a,b} = p_{a,b} \;\; \forall p_{a,b} \epsilon P^+$$

(3.8)

Αν δεν δίνονται προεπιλεγμένες τιμές για κανένα τετράγωνο του πλέγματος τότε το σύνολο P^+ είναι κενό και ο τρίτος περιορισμός δεν έχει κάποια επιρροή στο μοντέλο.

Έχοντας ορίσει τους περιορισμούς για τέσσερις από τους πέντε συνολικούς κανόνες του παιχνιδιού, απομένουν οι περιορισμοί για τον Κανόνα-4. Ο Κανόνας-4 προϋποθέτει να δοθούν κάποια στοιχεία για κάθε πλευρά του πλέγματος. Θεωρείται, όπως και προηγουμένως, ότι τα στοιχεία αυτά δίνονται στον πίνακα L (σχέση 3.9). Επειδή τώρα υπάρχουν μόνο 4 πλευρές σε κάθε πλέγμα και για κάθε πλευρά υπάρχουν N στοιχεία, ο πίνακας θα είναι 4xN:

$$L_{mat} = \begin{bmatrix} l_{1,1} & l_{1,2} & l_{1,3} & \ldots & l_{1,N} \\ l_{2,1} & l_{2,2} & l_{2,3} & \ldots & l_{2,N} \\ l_{3,1} & l_{3,2} & l_{3,3} & \ldots & l_{3,N} \\ l_{4,1} & l_{4,2} & l_{4,3} & \ldots & l_{4,N} \end{bmatrix}$$

$$l_{i,j} \epsilon [0, N] \;\; \forall i \epsilon [1, 4], \;\; \forall j \epsilon [1, N]$$

(3.9)

Η αντιστοιχία μεταξύ πλευράς και γραμμής του πίνακα L γίνεται ξεκινώντας από την πάνω πλευρά να αντιστοιχεί στην πρώτη γραμμή και συνεχίζοντας με δεξιόστροφη φορά:

- πρώτη γραμμή : πάνω πλευρά
- δεύτερη γραμμή : δεξιά πλευρά

- τρίτη γραμμή : κάτω πλευρά
- τέταρτη γραμμή : αριστερή πλευρά

Και σε αυτή την περίπτωση, ένα από τα στοιχεία που δίνονται μπορεί να είναι μηδέν. Όπως αναφέρθηκε στους κανόνες αυτό σημαίνει ότι το στοιχείο δεν δίνει κάποια πληροφορία και αγνοούμε αυτήν την τιμή. Αν ένα $l_{i,j}$ είναι μεγαλύτερο του μηδενός, τότε θα πρέπει να οριστεί ένας περιορισμός γι' αυτό, ο οποίος να επιβάλλει τον Κανόνα-4. Έτσι θα υπάρχουν μέχρι και 4N περιορισμοί για τον Κανόνα-4.

Η μορφή των περιορισμών αυτών θα εξαρτάται από την κατεύθυνση στην οποία "κοιτάζει" το στοιχείο. Αυτό σημαίνει ότι μόνο όσα στοιχεία είναι στην ίδια γραμμή στον πίνακα L (σχέση 3.9) θα μοιράζονται την ίδια μορφή περιορισμού. Ο λόγος που αλλάζει η μορφή του περιορισμού είναι γιατί κάποιοι από αυτούς αναφέρονται σε γραμμές και κάποιοι σε στήλες του πίνακα V (σχέση 3.1). Εξετάζουν την γραμμή/στήλη σαν μια λίστα από μεταβλητές, όπου παίζει ρόλο η σειρά. Η ορθή σειρά προσπέλασης των στοιχείων μιας γραμμής είναι από αριστερά προς τα δεξιά, πράγμα που αντιστοιχεί στην "ανατολική" κατεύθυνση. Αντίστοιχα η ορθή σειρά προσπέλασης μιας στήλης είναι από πάνω προς τα κάτω, άρα η "νότια" κατεύθυνση. Αλλά κάποια στοιχεία "βλέπουν" τις μεταβλητές στην ανάποδη κατεύθυνση, που θα ήταν η "δυτική" και "βόρεια" για γραμμές και στήλες αντίστοιχα.

Έτσι οι πιθανές μορφές των περιορισμών είναι:

- πάνω πλευρά (1η γραμμή του L) : βλέπει στήλες, ορθά
- δεξιά πλευρά (2η γραμμή του L) : βλέπει γραμμές, ανάποδα
- κάτω πλευρά (3η γραμμή του L) : βλέπει στήλες, ανάποδα
- αριστερή πλευρά (4η γραμμή του L) : βλέπει γραμμές, ορθά

Για να οριστούν οι τέσσερις αυτές μορφές των περιορισμών θα χρειαστούν μερικές βοηθητικές σχέσεις, καθώς οι περιορισμοί αυτοί είναι πολύ περίπλοκοι για να εκφραστούν ως έχειν.

Αρχικά οι σχέσεις που δίνουν μια γραμμή και μία στήλη του πίνακα V (σχέση 3.1) βάσει της θέσης της γραμμής/στήλης ορίζονται ως:

$$row_i = \{v_{i,j} \mid v_{i,j} \epsilon V \ \forall j \epsilon [1, N]\}$$

$$col_j = \{v_{i,j} \mid v_{i,j} \epsilon V \ \forall i \epsilon [1, N]\} \tag{3.10}$$

Το row_i δίνει την i-οστή γραμμή του πίνακα V (σχέση 3.1) σαν το σύνολο των μεταβλητών της. Η ορθή σειρά προσπέλασης των μεταβλητών αυτών βασίζεται στην μεταβλητή θέσης τους μέσα στην σειρά που είναι το j. Αφού κάθε $v_{i,j}$ στην rowi έχει το ίδιο i, το $v_{i,j}$ είναι πριν το $v_{i,j+1}$ (με εξαίρεση το $v_{i,N}$ που δεν ακολουθείται από καμία άλλη μεταβλητή). Αντίστοιχα το col_j αναφέρεται στην j-οστή γραμμή και παρομοίως η μεταβλητή θέσης είναι το i και το $v_{i,j}$ είναι πριν το $v_{i+1,j}$.

Για να μπορέσει να αποφανθεί το μοντέλο αν ένας ουρανοξύστης σε μία θέση, μιας λίστας μεταβλητών, είναι ορατός μπορεί να ακολουθήσει την παρακάτω λογική:

- Ο πρώτος ουρανοξύστης της λίστας είναι πάντα ορατός, διότι δεν υπάρχει προηγούμενος του να τον εμποδίσει.
- Ο δεύτερος ουρανοξύστης της λίστας θα είναι ορατός αν είναι ψηλότερος από τον πρώτο, έτσι ο πρώτος δεν τον κρύβει.

- Ο τρίτος ουρανοξύστης για να είναι ορατός πρέπει να μην εμποδίζεται ούτε από τον πρώτο ούτε από τον δεύτερο.
- Άρα για να είναι ορατός ένας ουρανοξύστης στην θέση x της λίστας (x>1) πρέπει να είναι μεγαλύτερος από όλους τους προηγούμενους του.
- Αυτό συνεπάγεται ότι ένας ουρανοξύστης είναι ορατός αν και μόνο αν είναι ο υψηλότερος στο υποσύνολο που περιλαμβάνει αυτόν και όλους τους προηγούμενους του.

Για να μοντελοποιηθεί αυτό μαθηματικά ορίζεται η συνάρτηση $visible_i(j)$ για τις γραμμές και $visible_j(i)$ για τις στήλες:

$$visible_i(j) = \begin{cases} 1 & if \; v_{i,j} > v_{i,k} \; \forall k < j \\ 0 & otherwise \end{cases}$$

$$visible_j(i) = \begin{cases} 1 & if \; v_{i,j} > v_{k,j} \; \forall k < i \\ 0 & otherwise \end{cases}$$

(3.11)

Οι συναντήσεις αυτές παίρνουν την τιμή 1 όταν ο ουρανοξύστης είναι ορατός και την τιμή 0 όταν κρύβεται από κάποιον προηγούμενο του. Ακολουθούν την λογική που παρατέθηκε παραπάνω. Αλλά έχουν οριστεί μόνο για την ορθή προσπέλαση της γραμμής/στήλης, άρα πρέπει να οριστούν και οι συναρτήσεις για τις ανάποδες προσπελάσεις, αυτές είναι οι $visible_{-i}(j)$ και $visible_{-j}(i)$:

$$visible_i^-(j) = \begin{cases} 1 & if \; v_{i,j} > v_{i,k} \; \forall k > j \\ 0 & otherwise \end{cases}$$

$$visible_j^-(i) = \begin{cases} 1 & if \; v_{i,j} > v_{k,j} \; \forall k > i \\ 0 & otherwise \end{cases}$$

(3.12)

Η ανάποδη προσπέλαση εδώ παίρνει την μορφή της εξέτασης του εν λόγω ουρανοξύστη και όλων των επομένων του στην λίστα σαν υποσύνολο (αντί για όλων των προηγουμένων του).

Χάρις αυτές τις τέσσερις συναρτήσεις καλύπτονται οι τέσσερις μορφές περιορισμών που προαναφέρθηκαν. Τώρα για κάθε στοιχείο του πίνακα L (σχέση 3.9) μπορεί να οριστεί η αντίστοιχη μορφή περιορισμού:

$$C_{1,j}^l : l_{1,j} = \sum_{i=1}^{N} visible_j(i)$$

$$C_{2,i}^l : l_{2,i} = \sum_{j=1}^{N} visible_i^-(j)$$

$$C_{3,j}^l : l_{3,j} = \sum_{i=1}^{N} visible_j^-(i)$$

$$C^l_{4,i} : l_{4,i} = \sum_{j=1}^{N} visible_i(j)$$

(3.14)

Η κάθε μία από αυτές τις μορφές περιορισμού $C^l_{i,j}$ αντιστοιχεί στο στοιχείο $l_{i,j}$ του πίνακα L. Και ορίζει τον περιορισμό ότι η αντίστοιχη γραμμή/στήλη πρέπει να έχει αριθμό ορατών ουρανοξυστών ίσο με το στοιχείο κοιτώντας στην σωστή κατεύθυνση. Το καταφέρνουν αυτό καλώντας την αντίστοιχα σωστή συνάρτηση για όλα τα στοιχεία της γραμμής/στήλης. Δημιουργεί έτσι ένα σύνολο όπου τα στοιχεία του αντιστοιχούν το καθένα σε έναν ουρανοξύστη της λίστας και είναι 0 ή 1 ανάλογα με το αν είναι ορατοί. Παίρνοντας το άθροισμα των στοιχείων του συνόλου αυτού ο περιορισμός μπορεί να αποφανθεί για το αν είναι έγκυρη η λύση.

Κλείνοντας, το σύνολο όλων των περιορισμών για κάθε δοσμένο πρόβλημα ορίζεται από την σχέση 3.15:

$$C = \{C_1,\ C_2,\ C_3\} \bigcup \{C^l_{i,j} \mid l_{i,j} \neq 0,\ \forall l_{i,j} \epsilon L\}$$

(3.15)

3.4 Προγραμματιστικό μοντέλο

Το μαθηματικό μοντέλο αποτελεί μια πλήρης και μαθηματικά ακριβής εικόνα του παιχνιδιού, αλλά δεν δίνει λύση στο πρόβλημα. Για να επιλυθεί το πρόβλημα με αλγοριθμικό τρόπο πρέπει να χρησιμοποιηθεί προγραμματισμός ικανοποίησης περιορισμών. Σκοπός ενός αλγοριθμικού μοντέλου είναι να επιδείξει μια μέθοδο, ακολουθώντας την οποία ένας υπολογιστής μπορεί να λύσει συστηματικά οποιοδήποτε δοσμένο πρόβλημα τύπου “skyscrapers”. Εδώ ο όρος “προγραμματισμός” δεν αναφέρεται στην δημιουργία ενός προγράμματος, αλλά στην υλοποίηση ενός σχεδίου βάση του οποίου θα φτιαχτεί το αλγοριθμικό μοντέλο.

Σαν λύση του προβλήματος ορίζεται μια απόδοση τιμών σε όλες τις μεταβλητές του προβλήματος. Η εκάστοτε καθορισμένη τιμή μιας μεταβλητής πρέπει να είναι μια που ανήκει στο αντίστοιχο πεδίο ορισμού της μεταβλητής· πρέπει επίσης να μην παραβιάζεται κανένας περιορισμός. Αυτή η ανάθεση τιμών είναι μια εφικτή λύση. Σε αυτή την περίπτωση η εφικτή λύση αρκεί, καθώς ο παίκτης κερδίζει το παιχνίδι όταν συμπληρώσει το πλέγμα, δεν υπάρχει κάποιο σύστημα βαθμολόγησης που να δίνει βαθμό (score) σε μια λύση. Αυτό σημαίνει πως δεν υπάρχει κάποια αντικειμενική συνάρτηση που θα χρειαζόταν να βελτιστοποιήσει το προγραμματιστικό μοντέλο.

Για να λύσει ένα πρόβλημα το αλγοριθμικό μοντέλο πρέπει να δοκιμάσει όλες τις πιθανές αναθέσεις τιμών σε μεταβλητές μέχρι να βρει μια που ικανοποιεί τους περιορισμούς. Τυπικά αυτό γίνεται μέσω της προσπέλασης του δέντρου καταστάσεων, κάνοντας αναζήτηση σε δέντρο. Άρα ο αλγόριθμος κάνει αναζήτηση για να βρει κάποιο από τα φύλλα του δέντρου που αποτελεί λύση. Για να μειωθεί ο χώρος αναζήτησης, με αποτέλεσμα να γίνει ταχύτερος ο αλγόριθμος, μπορούν να επιστρατευτούν οι τεχνικές του προγραμματισμού ικανοποίησης περιορισμών, που παρουσιάστηκαν στο κεφάλαιο 1.

Πριν να υπάρξει η δυνατότητα να προγραμματιστεί η λύση ενός γρίφου “skyscrapers” από τον υπολογιστή πρέπει να αναλυθούν οι περιορισμοί του συνόλου C (12). Μέσα από αυτή την ανάλυση θα βρεθεί ο βέλτιστος τρόπος να χρησιμοποιηθούν οι τεχνικές επίλυσης, για να επιταχυνθεί όσο είναι δυνατόν η λύση του. Έτσι θα δοθεί ένα θεωρητικό και γενικό (ανεξάρτητο από τεχνικές λεπτομέρειες υλοποίησης) σχέδιο για την υλοποίηση ενός προγράμματος που θα λύνει το πρόβλημα.

Το αρχικό σύνολο ασυμπλήρωτων μεταβλητών (unlabeled) είναι όλες οι μεταβλητές. Σύμφωνα με τον περιορισμό C_3 (7) μπορούν να συμπληρωθούν αμέσως όσες μεταβλητές των οποίων τα πλαίσια δίνονται συμπληρωμένα εξ’ αρχής. Αυτό είναι αντίστοιχο της εφαρμογής του NC για τα τον C_3 που είναι μοναδικός περιορισμός. Δίνονται οι αντίστοιχες τιμές στις μεταβλητές και αυτές αφαιρούνται από το σύνολο ασυμπλήρωτων, με αυτή την κατάσταση θα αρχίσει η αναζήτηση. Από αυτό το σημείο και μετά, ο περιορισμός C_3 (7) δεν συμμετέχει καθόλου στην αναζήτηση, καθώς έχει ήδη ικανοποιηθεί.

Εφόσον πρόκειται για στάδιο προ-επεξεργασίας (η αναζήτηση δεν έχει αρχίσει ακόμα) εδώ μπορεί να εφαρμοστεί και κάποια ακόμη τεχνική συνέπειας. Η PC θα μπορούσε να έχει εφαρμογή εδώ και να αφαιρέσει κάποιες ασυνεπείς τιμές, προτείνεται η χρήση τους όμως μόνο για τους περιορισμούς C_1 (4) και C_2 (5). Οι C_1 και C_2 είναι ήδη δυαδικοί περιορισμοί σε αντίθεση με τους C_l (11) που χρειάζονται δυαδικοποίηση. Η διαδικοποίηση είναι μία διαδικασία που, όπως έχει αναφερθεί, είναι ακριβή σε πόρους και είναι προτιμότερο να αποφεύγεται για τις τεχνικές συνέπειας.

Ο λόγος που επιλέχθηκε η PC σαν τεχνική συνέπειας, αντί να επιλεγεί η AC, είναι διότι στους περιορισμούς C_1 (4) και C_2 (5) το μέγιστο μέγεθος μονοπατιού που υπάρχει στο γράφημα περιορισμών είναι γνωστό. Οι περιορισμοί ελέγχουν ότι οι μεταβλητές της ίδιας γραμμής/στήλης δεν έχουν τις ίδιες τιμές. Αφού μία γραμμή/στήλη έχει μήκος N τότε το μέγιστο μονοπάτι μεταξύ δύο μεταβλητών είναι μήκους N-1. Επειδή το μήκος των μονοπατιών που θα πρέπει να ελεγχθούν είναι γνωστό, είναι πιο εύκολη η επιλογή του άνω ορίου μήκους που θα τεθεί γι’ αυτή την μέθοδο.

Περνώντας από το προκαταρκτικό στάδιο των τεχνικών συνέπειας στην αναζήτηση, θα εξεταστεί το ενδεχόμενο χρήσης κάποιας τεχνικής διάταξης μεταβλητών και τιμών. Δυστυχώς δεν γίνεται να επιλεγεί κάποια από τις μεθόδους με θεωρητικό τρόπο. Η επιλογή θα γίνει αργότερα στο πρακτικό κομμάτι μέσω πειραμάτων.

Για τις τεχνικές οπισθοχώρησης και προώθησης περιορισμών οι Backjumping και FC μπορούν να υιοθετηθούν. Εξετάζοντας τους περιορισμούς C_1 (4) και C_2 (5) η προώθηση τους γίνεται με απλό τρόπο. Όταν μια μεταβλητή παίρνει τιμή ο αλγόριθμος πρέπει στην ίδια γραμμή να αφαιρέσει την τιμή αυτή από το πεδίο ορισμού όλων των άλλων μεταβλητών. Αντίστοιχα πρέπει να γίνει το ίδιο και για όλες τις άλλες μεταβλητές στην ίδια στήλη. Με αυτόν τον τρόπο εξασφαλίζεται ότι δεν θα εξεταστεί μη έγκυρη τιμή για αυτές τις μεταβλητές όταν θα τους ανατεθεί τιμή. Ακόμη η μεταβλητή στην οποία ανατέθηκε τιμή θα μπει στο σύνολο συγκρούσεων όλων των μεταβλητών της ίδιας γραμμής και ίδιας στήλης. Με αυτόν τον τρόπο θα μπορεί ο αλγόριθμος να κάνει στρατηγική οπισθοδρόμηση στη

τελευταία μεταβλητή που προκάλεσε την τρέχουσα να μην έχει διαθέσιμες τιμές για ανάθεση στο πεδίο ορισμού της. Φυσικά όταν γίνεται μια οπισθοδρόμηση οι αλλαγές στα πεδία ορισμού και σύνολα συγκρούσεων των άλλων μεταβλητών αναιρούνται. Η μόνη αλλαγή που παραμένει είναι η αφαίρεση της τιμής που προκάλεσε το αδιέξοδο από το πεδίο ορισμού της μεταβλητής στην οποία επέστρεψε ο αλγόριθμος.

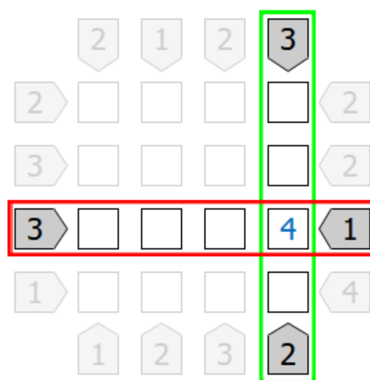
Για τους περιορισμούς C_l (11) η προώθηση απαιτεί μεγαλύτερη διαδικασία. Εδώ θα δοθεί μια μέθοδος για να γίνει προώθηση των C^l. Επειδή η μέθοδος αυτή απαιτεί έλεγχο πολλαπλών συνδυασμών, υπάρχει η περίπτωση να μην είναι αρκετά αποδοτική. Η εναλλακτική επιλογή θα ήταν να προωθηθούν μόνο οι περιορισμοί C_1 (4) και C_2 (5), αλλά αυτό θα έχει ως αποτέλεσμα να συναντήσει ο αλγόριθμος περισσότερα αδιέξοδα στην αναζήτηση. Στο πρακτικό κομμάτι θα ελεγχθεί πειραματικά αν η προώθηση των Cl είναι αποδοτική.

Όταν ανατίθεται μια τιμή σε μια μεταβλητή θα πρέπει να διαγραφούν οι ασυνεπείς τιμές, σύμφωνα με τον C^l (11), από τα πεδία ορισμού των μεταβλητών που βρίσκονται στην ίδια γραμμή και στήλη με την τρέχουσα μεταβλητή. Θα πρέπει να εξεταστούν τέσσερα το πολύ υποσύνολα, ένα από κάθε κατεύθυνση στην οποία υπάρχει στοιχείο. Για το κάθε υποσύνολο θα πρέπει να εξεταστούν όλοι οι πιθανοί συνδυασμοί τιμών για τις ασυμπλήρωτες μεταβλητές και να διαγραφούν όσες δεν τηρούν κάποιον από τους περιορισμούς. Για παράδειγμα αν το μοντέλο εξετάζει την μεταβλητή του σχήματος 3.6:

Σχήμα 33.21: Επιλογή μεταβλητής.

Το μοντέλο έχει επιλέξει την μεταβλητή v3,4 και την έχει συμπληρώσει με 4.

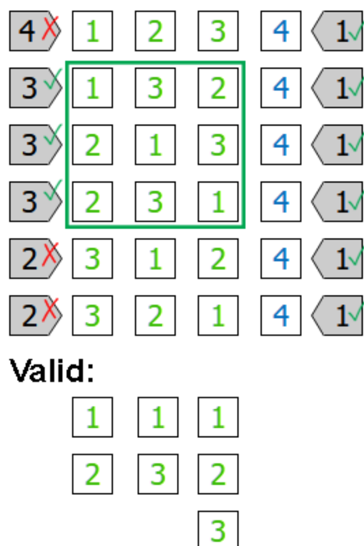
Θα πρέπει να ελέγξει τα παρακάτω δύο υποσύνολα των μεταβλητών, η γραμμή και η στήλη στις οποίες ανήκει η μεταβλητή, που φαίνονται στο σχήμα 3.7:



Σχήμα 33.22: Η γραμμή και η στήλη της επιλεγμένης μεταβλητή.

Το κάθε υποσύνολο θα ελεγχθεί ορθά και ανάποδα ανάλογα με το ποια από τα τέσσερα στοιχεία πλευράς δίνονται γι' αυτή την μεταβλητή. Για το κάθε υποσύνολο θα πρέπει να παραχθούν όλοι οι συνδυασμοί τιμών για τις μεταβλητές που δεν έχουν ήδη συμπληρωθεί σε προηγούμενο βήμα της αναζήτησης. Για κάθε πιθανό συνδυασμό θα πρέπει να ελέγξει αν ισχύει ο αντίστοιχος περιορισμός C^l (11) και να επιτρέψει στην μεταβλητή να κρατήσει τις τιμές για τις οποίες ισχύει, στο πεδίο ορισμού της.

Στη γραμμή της μεταβλητής του παραπάνω παραδείγματος θα παρήγαγε την παρακάτω προώθηση περιορισμών:



Σχήμα 33.23: Προώθηση περιορισμών.

Το μοντέλο ελέγχει όλες τις πιθανές αναθέσεις για τις μεταβλητές αυτής της γραμμής. Θέτει σαν έγκυρες τις αναθέσεις που ικανοποιούν τα στοιχεία και διαγράφει τις μη έγκυρες από τα πεδία ορισμών των μεταβλητών.

Στην μέση φαίνονται όλοι οι πιθανοί συνδυασμοί των ασυμπλήρωτων τιμών και δεξιά και αριστερά ο υπολογισμένος αριθμός από ορατούς ουρανοξύστες. Προφανώς μόνο οι συνδυασμοί που έχουν σωστό

αριθμό από ορατούς ουρανοξύστες αριστερά (3) και δεξιά (1) είναι αποδεκτοί. Από τους αποδεκτούς συνδυασμούς επιλέγονται οι συνεπείς τιμές της κάθε ασυμπλήρωτης μεταβλητής. Αρκεί μια τιμή να υπάρχει σε έναν αποδεκτό συνδυασμό για να είναι συνεπής, για παράδειγμα η πρώτη μεταβλητή της παραπάνω γραμμής έχει συνεπείς τιμές τις (1,2). Οι τιμές που δεν είναι συνεπείς μπορούν αφαιρεθούν, για την πρώτη μεταβλητή της παραπάνω γραμμής οι (3,4).

Ο λόγος που απαιτείται υπολογισμός όλων αυτών των συνδυασμών, είναι διότι είναι αδύνατο να αποφανθεί ο αλγόριθμος για όλες τις πιθανές τιμές μιας μεταβλητής, ότι είναι συνεπείς γνωρίζοντας μόνο τις τιμές των συμπληρωμένων. Η συνέπεια μιας τιμής εξαρτάται από τις τιμές που θα πάρουν και οι υπόλοιπες ασυμπλήρωτες. Γι' αυτό τον λόγο είναι αναγκαίο, ο αλγόριθμος να δοκιμάσει όλες τις δυνατές πιθανότητες.

3.5 Υλοποίηση προγραμματιστικού μοντέλου

Για να ελεγχθεί η ορθότητα και απόδοση του προγραμματιστικού μοντέλου αυτό υλοποιήθηκε σε Python με χρήση της βιβλιοθήκης OR-Tools (Operational Research Tools) της Google [61]. Τα OR-Tools είναι μια βιβλιοθήκη ανοιχτού κώδικα (open-source), η οποία φτιάχτηκε με σκοπό την επίλυση προβλημάτων βελτιστοποίησης, δρομολόγησης οχημάτων, χρονοπρογραμματισμού, ακεραίου και γραμμικού προγραμματισμού (integer, linear programming) και προγραμματισμού περιορισμών. Η βιβλιοθήκη διατίθεται για τις γλώσσες C++, Python, C# και Java.

Τα Or-Tools περιλαμβάνουν το πακέτο CP-SAT solver το οποίο είναι φτιαγμένο για να λύνει προβλήματα ικανοποίησης περιορισμών χρησιμοποιώντας τεχνικές προβλημάτων SAT (satisfiability). Το πακέτο CP-SAT περιλαμβάνει ένα API (application programming interface) για να διευκολύνει τον χρήστη στο να υλοποιήσει το μαθηματικό μοντέλο του και να βρει μια λύση. Παρότι το API είναι περιοριστικό, το πακέτο αναλαμβάνει μόνο του την λύση, προώθηση περιορισμών, οπισθοδρομήσεις και μεθόδους επιτάχυνσης της αναζήτησης. Χρησιμοποιώντας το API ο χρήστης μπορεί με μικρό κόστος σε χρόνο, να υλοποιήσει ένα πρόγραμμα που θα λύσει το πρόβλημα του.

Σε αυτό το σημείο θα παρουσιαστεί η χρήση των Or-Tools μέσω της επίλυσης του προβλήματος "Skyscrapers". Το πρόγραμμα που λύνει έναν γρίφο Skyscrapers υλοποιήθηκε στο αρχείο skyscrapers_solver.py. Ο κώδικας του φαίνεται παρακάτω (ο πλήρης κώδικας παρατίθεται στο Παράρτημα Α):

```
grid_size = 4                                   # NxN grid size
domain = (1, grid_size)                         # domain for all the
variables
preplaced = []                                  # preplaced values
constraint
top     = [1,2,3,2]                             # four sides constraints
right   = [2,3,1,2]
bottom  = [3,2,1,2]
left    = [1,2,2,3]

model = cp_model.CpModel()                      # make model object
solver = cp_model.CpSolver()                    # make solver object
```

```
set_variables(grid_size, model, domain)       # generate the variables
set_constraints(grid_size, model, preplaced, # generate the
constraints
                top, right, bottom, left)

status = solver.Solve(model)                  # solve the problem
```

Ο χρήστης πρέπει αρχικά να θέσει τις παραμέτρους του τρέχοντα γρίφου. Αυτές είναι το grid_size που είναι το μέγεθος του πλέγματος (N), η λίστα preplaced που είναι τα τετράγωνα που δίνονται εξ αρχής στον γρίφο (συμφωνά με τον κανόνα-5) και οι λίστες top, right, bottom, left που είναι τα στοιχεία της κάθε πλευράς (συμφωνά με τον κανόνα-4). Η λίστα preplaced συμπληρώνεται με τον ίδιο τρόπο που φτιάχνεται το σύνολο P^+ (3.7). Αν δίνεται ότι ο ουρανοξύστης στην θέση (i,j) έχει ύψος h τότε η πλειάδα (i, j, h) πρέπει να ενταχθεί στη λίστα preplaced πριν την εκτέλεση του προγράμματος. Αν δεν δίνεται κανένα στοιχείο τέτοιου τύπου η λίστα preplaced μένει κενή. Οι τέσσερις λίστες στοιχείων πλευράς είναι αντίστοιχες με τις τέσσερις γραμμές του πίνακα L (3.9), τα στοιχεία δίνονται από δεξιά προς τα αριστερά για τις οριζόντιες πλευρές του πλέγματος (top και bottom) και από πάνω προς τα κάτω για τις κάθετες πλευρές (right και left). Αν κάποιο στοιχείο δεν δίνεται τότε πρέπει να αντικαθίσταται από 0.

Αφού συμπληρωθούν οι παράμετροι του γρίφου προς επίλυση, το πρόγραμμα φτιάχνει δύο αντικείμενα, ένα CpModel και ένα CpSolver. Το αντικείμενο CpModel χρησιμοποιείται για να υλοποιήσει ο χρήστης ένα μαθηματικό μοντέλο. Αυτό γίνεται προσθέτοντας μεταβλητές και περιορισμούς στο μοντέλο. Οι μεταβλητές προστίθενται με την μέθοδο *set_variables* ενώ οι περιορισμοί με την μέθοδο *set_constraints*. Έπειτα το CpSolver χρησιμοποιείται για να εκτελέσει την αναζήτηση και τυπώνει την λύση που βρίσκει, αν υπάρχει.

Η μέθοδος set_variables χειρίζεται την δημιουργία των μεταβλητών του μοντέλου λειτουργεί με τον παρακάτω κώδικα:

```
for i in range(grid_size):
        variables.append([])
        for j in range(grid_size):
            var = model.NewIntVar(domain[0], domain[1],
                                  'v'+str(i)+','+str(j))
            variables[-1].append(var)
        V = np.array(variables)
```

Με ένα διπλό βρόγχο φτιάχνει τον πίνακα *variables* ο οποίος είναι αντίστοιχος του V (3.1) αποθηκεύει τις μεταβλητές στις θέσεις του πίνακα που είναι αντίστοιχες με τις θέσεις του πλέγματος. Οι μεταβλητές του προβλήματος Skyscrapers μπορούν να πάρουν μόνο ακέραιες τιμές, γι' αυτό χρησιμοποιούνται αντικείμενα τύπου IntVar από το πακέτο CP-SAT. Καλώντας την μέθοδο CpModel.NewIntVar το αντικείμενο της μεταβλητής δημιουργείται και προστίθεται στο μοντέλο ταυτόχρονα. Για να κληθεί η μέθοδος δίνονται σαν ορίσματα τα άνω και κάτω όρια του πεδίου ορισμού της μεταβλητής καθώς και ένα όνομα γι' αυτήν, το όνομα είναι "vi,j" για την μεταβλητή στην

θέση (i,j) του πίνακα. Ο πίνακας μετατρέπεται σε αντικείμενο τύπου NumPy array [68] το οποίο θα βοηθήσει στην εύκολη δημιουργία των περιορισμών.

Έπειτα η μέθοδος *set_constraints* θα προσθέσει τους περιορισμούς στο μοντέλο. Αρχικά προστίθενται οι $\mathbf{C}_1$ (4) και $\mathbf{C}_2$ (5):

```
for i in range(grid_size):
        model.AddAllDifferent(V[i,:].tolist())
        model.AddAllDifferent(V[:,i].tolist())
```

Με την χρήση της μεθόδου CpModel.AddAllDifferent μπορούν πολύ εύκολα να προστεθεί ένας περιορισμός που ορίζει ότι σε μία λίστα μεταβλητών πρέπει να έχουν όλες διαφορετική τιμή. Επειδή αυτός είναι ένας συχνός περιορισμός στα CSP η μέθοδος έχει ήδη υλοποιηθεί για να επιταχύνει την ανάπτυξη των προγραμμάτων. Εδώ χρησιμοποιείται και η δυνατότητα των NumPy arrays που επιτρέπει εύκολη διαμέριση (slicing), μία απλή δισδιάστατη λίστα της Python δεν έχει δυνατότητα προσπέλασης στήλης προς στήλη.

Αν υπάρχουν στοιχεία για τον περιορισμό C_3 (7), αυτά προστίθεται ως εξής:

```
for pre in preplaced:
        model.AddHint(V[pre[0],pre[1]], pre[2])
```

Η μέθοδος CpModel.AddHint δέχεται σαν ορίσματα μια μεταβλητή και μια τιμή και δίνει την τιμή αυτή στην μεταβλητή πριν να ξεκινήσει η αναζήτηση, η μεταβλητή θεωρείται συμπληρωμένη.

Απομένουν να προστεθούν οι περιορισμοί C^l (11) στο μοντέλο. Ο λογικός τρόπος που θα μπορούσαν να προστεθούν είναι χρησιμοποιώντας μια συνάρτηση:

```
def visible(row):
    seen = 1
    cur_max = row[0]
    for scr in row[1:]:
        if scr > cur_max:
            seen += 1
            cur_max = scr
    return seen
```

Η συνάρτηση visible μετράει για μια γραμμή μεταβλητών πόσες είναι ορατές στην κατεύθυνση όπου αυξάνεται το index της λίστας. Έπειτα θα αρκούσε να ελεγχθεί ότι το στοιχείο σε εκείνη την κατεύθυνση είναι ίσο με τους ουρανοξύστες που είναι ορατοί:

```
model.Add(visible(V[i,:])) == left[i])
```

Αυτού του είδους ο ορισμός όμως είναι λάθος, για δύο λόγους: πρώτον η αποτίμηση της συνάρτησης visible είναι αδύνατη αν η γραμμή/στήλη περιέχει ασυμπλήρωτες μεταβλητές. Δεύτερον το API του CP-SAT δεν επιτρέπει, κατά κύριο λόγο, την χρήση πολύπλοκων εκφράσεων στους περιορισμούς. Η μέθοδος CpModel.Add προσθέτει έναν περιορισμό στο μοντέλο, αυτός ο περιορισμός πρέπει να είναι της μορφής οριοθετημένης γραμμικής έκφρασης (BoundedLinearExpression), η οποία επιτρέπει μόνο αριθμητικές πράξεις ακεραίων και ανισώσεις μεταξύ των μεταβλητών.

Λόγο της περιοριστικής φύσης του API των OR-Tools και της πολυπλοκότητας των περιορισμών C^l (11), πρέπει αυτοί να οριστούν αλλιώς. Υπάρχουν δύο τρόποι να γίνει αυτό: ο πρώτος είναι να μην προστεθούν στο μοντέλο οι C^l, έτσι το μοντέλο θα δίνει λύσεις που τηρούν μόνο τους περιορισμούς C_1 (4), C_2 (5) και C_3 (7). Παίρνοντας μια από αυτές τις λύσεις, το πρόγραμμα (με χρήση απλού κώδικα, όχι των OR-Tools) θα ελέγχει αν ικανοποιεί τους C^l και αν δεν τους ικανοποιεί τότε ζητάει την επόμενη λύση.

Ο δεύτερος τρόπος προϋποθέτει να δημιουργηθεί ένα λεξικό που να περιέχει σαν τιμές όλες τις πιθανές μεταθέσεις N αριθμών. Αυτές είναι δηλαδή όλες οι πιθανές συμπληρωμένες γραμμές και στήλες που θα μπορούσαν να υπάρχουν σε ένα πλέγμα NxN. Κλειδί θα είναι μια πλειάδα που περιέχει τους αριθμούς των ουρανοξυστών που είναι ορατοί από, αριστερά και δεξιά. Παράδειγμα του λεξικού που παράγεται για N=4 βρίσκεται στο σχήμα 3.9.

```
(4, 1)
        (1, 2, 3, 4)
(3, 2)
        (1, 2, 4, 3)
        (1, 3, 4, 2)
        (2, 3, 4, 1)
(3, 1)
        (1, 3, 2, 4)
        (2, 1, 3, 4)
        (2, 3, 1, 4)
(2, 2)
        (1, 4, 2, 3)
        (2, 1, 4, 3)
        (2, 4, 1, 3)
        (3, 1, 4, 2)
        (3, 2, 4, 1)
        (3, 4, 1, 2)
(2, 3)
        (1, 4, 3, 2)
        (2, 4, 3, 1)
        (3, 4, 2, 1)
(2, 1)
        (3, 1, 2, 4)
        (3, 2, 1, 4)
(1, 2)
        (4, 1, 2, 3)
        (4, 2, 1, 3)
(1, 3)
        (4, 1, 3, 2)
        (4, 2, 3, 1)
        (4, 3, 1, 2)
(1, 4)
        (4, 3, 2, 1)
```

Σχήμα 33.24: Παράδειγμα του λεξικού για N=4.

Αν έπρεπε να βρεθεί ποιοι πιθανοί συνδυασμοί επιτρέπονται έτσι ώστε συμπληρωθεί η γραμμή του σχήματος 3.4, αυτό μπορεί να γίνει παίρνοντας την τιμή (3,2) (αυτά είναι τα στοιχεία αριστερά και δεξιά της γραμμής) από το λεξικό.

Χρησιμοποιώντας αυτό το λεξικό ένας περιορισμός C^l (11) προστίθεται εύκολα στο μοντέλο:

```
model.AddAllowedAssignments(V[idx,:].tolist(), assignments[key])
```

Για ένα ζευγάρι περιορισμών που βρίσκονται αριστερά και δεξιά μιας γραμμής (ή πάνω και κάτω μιας στήλης) αρχικά φτιάχνεται το κλειδί *key* και μετά προστίθενται οι επιτρεπτοί περιορισμοί από το λεξικό *assignments* με την χρήση της CpModel.AddAllowedAssignments. Η μέθοδος αυτή προσθέτει έναν περιορισμό επιτρεπτών συνδυασμών αναθέσεών. Πιο συγκεκριμένα δέχεται σαν είσοδο μία λίστα από μεταβλητές και μια λίστα από πλειάδες που η κάθε μία πλειάδα περιέχει έναν επιτρεπτό συνδυασμό από αναθέσεις γι’ αυτές τις μεταβλητές. Πρέπει το παραπάνω να γίνει για όλα τα ζευγάρια στοιχείων αριστερής-δεξιάς πλευράς και πάνω-κάτω πλευράς. Ακόμη αν ένα από τα δύο στοιχεία του κλειδιού λείπει (δίνεται ως 0) πρέπει στους επιτρεπτούς συνδυασμούς αυτής της γραμμής/στήλης να προστεθούν οι επιτρεπτοί συνδυασμοί από όλες τις τιμές του *assignments* που έχουν το μη μηδενικό μέρος του κλειδιού. Αν για παράδειγμα τα στοιχεία είναι (0,2) πρέπει στους επιτρεπτούς συνδυασμούς να προστεθούν οι λίστες των κλειδιών: (1,2), (2,2), (3,2). Αν δεν υπάρχουν στοιχεία και στις δύο πλευρές τότε όλοι οι συνδυασμοί είναι επιτρεπτοί για αυτήν την γραμμή/στήλη και δεν χρειάζεται να δημιουργηθεί περιορισμός.

Πίνακας 33.2: Επιλογές OR-Tools για στρατηγικές επιλογής μεταβλητής-τιμής [69]

Type	Strategy	Description
Variable	CHOOSE_FIRST	The first not-fixed variable in the list.
	CHOOSE_HIGHEST_MAX	The variable that could (potentially) take the highest value.
	CHOOSE_LOWEST_MIN	The variable that could (potentially) take the lowest value.
	CHOOSE_MAX_DOMAIN_SIZE	The variable that has the most feasible assignments.
	CHOOSE_MIN_DOMAIN_SIZE	The variable that has the fewest feasible assignments.
Value	SELECT_LOWER_HALF	Branch to the lower half.
	SELECT_MAX_VALUE	Try to assign the largest value.
	SELECT_MIN_VALUE	Try to assign the smallest value.
	SELECT_UPPER_HALF	Branch to the upper half.

Πέρα από τον ορισμό των μεταβλητών και των περιορισμών το CP-SAT δεν επιτρέπει παραπάνω έλεγχο της αναζήτησης από τον χρήστη. Η οπισθοδρόμηση και προώθηση περιορισμών γίνεται αυτόματα χωρίς πολλές επιλογές παραμετροποίησης. Κάτι στο οποίο έχει επιλογή ο χρήστης είναι οι

ευρετικές συναρτήσεις επιλογής επόμενης μεταβλητής και τιμής σε κάθε βήμα, μέσω της συνάρτησης CpModel.AddDecisionStrategy. Η συνάρτηση αυτή παίρνει σαν ορίσματα μία λίστα από μεταβλητές και δύο ακέραιους που επιλέγουν μία στρατηγική επιλογής επόμενης μεταβλητής και μία στρατηγική επιλογής επόμενης τιμής. Οι στρατηγικές αυτές είναι ορισμένες στο CP-SAT και δεν αλλάζουν, ούτε υπάρχει η δυνατότητα να προστεθεί κάποια νέα στρατηγική.

Αφού τεθούν οι μεταβλητές και οι περιορισμοί, ο χρήστης μπορεί να χρησιμοποιήσει την μέθοδο Solve του CpSolver για να λύσει το μοντέλο. Σε αυτό το σημείο αξίζει να αναφερθεί και η κλάση CpSolverSolutionCallback που περιέχει το πακέτο CP-SAT. Υλοποιώντας την ο χρήστης μπορεί να φτιάξει μία συνάρτηση η οποία θα κληθεί όταν ο CpSolver βρει μια λύση και μπορεί να χρησιμοποιηθεί για να τυπωθεί η λύση και ίσως να γίνει κάποια απόφαση για το αν το μοντέλο θα συνεχίσει την αναζήτηση ή όχι. Όταν η αναζήτηση τελειώσει η μέθοδος Solve θα επιστρέψει έναν κωδικό κατάστασης: OPTIMAL και FEASIBLE για επιτυχής λύση (δεν υπάρχει βελτιστοποίηση σε αυτό το πρόβλημα άρα οι δύο κωδικοί αυτοί είναι ισοδύναμοι), INFEASIBLE αν αποδεδειγμένα καταλήγει μόνο σε αδιέξοδα και MODEL_INVALID ή UNKNOWN για σφάλματα.

```
-------------
   1 2 3 2
   V V V V
1> 4 1 2 3 <2
2> 1 4 3 2 <3
2> 3 2 1 4 <1
3> 2 3 4 1 <2
   Λ Λ Λ Λ
   3 2 1 2
-------------
time: 0.010387420654296875
FEASIBLE, 1 solutions
```

Σχήμα 33.25: Έξοδος του προγράμματος skyscrapers_solver.py.

Το πρόγραμμα skyscrapers_tester.py (ο πλήρης κώδικας παρατίθεται στο Παράρτημα Β) δοκιμάζει και μια δεύτερη εκδοχή επίλυσης του προβλήματος. Αντί να εκτελεί τον κώδικα της gen_perm_dict, για να υπολογίσει όλες τις μεταθέσεις μιας λίστας και τα στοιχεία τα οποία θα αντιστοιχούσαν σε κάθε μια, δεν βάζει τους περιορισμούς C^1 (σχέση 3.14) στο μοντέλο. Αντί να αφήσει το μοντέλο να διαχειριστεί τους περιορισμούς μόνο του, το χρησιμοποιεί για να πάρει λύσεις όπου ισχύουν οι περιορισμοί C1 και C2 (σχέσεις 3.4 και 3.5) και μετά ελέγχει αν ισχύουν οι περιορισμοί C^1 στην CpSolverSolutionCallback. Αν είναι αποδεκτή λύση τότε σταματάει την αναζήτηση, αλλιώς την αφήνει να συνεχίσει.

Για να αποφασίσει ο χρήστης αν θα χρησιμοποιήσει την προώθηση ή όχι θέτει μια μεταβλητή forward σε True ή False στην αρχή του προγράμματος. Ακόμη θέτει μια μεταβλητή τύπου range για να επιλέξει το εύρος του μεγέθους πλέγματος το οποίο θα δοκιμάσει. Σε αυτό το πρόγραμμα το εύρος μπορεί να πάει από 2 έως 11. Για κάθε μέγεθος υπάρχει στο πρόγραμμα μια λίστα που περιέχει τα στοιχεία για έναν έτοιμο επιλύσιμο γρίφο τον οποίο θα λύσει το μοντέλο. Για να υπάρχει τυχαιότητα το πρόγραμμα κάνει τέσσερα πειράματα, περιστρέφοντας κάθε φορά τα στοιχεία αντίθετα με την φορά του ρολογιού. Αυτό δημιουργεί έναν καινούργιο γρίφο ο οποίος είναι επίσης επιλύσιμος. Έτσι είναι σίγουρο πως το μοντέλο δεν τυχαίνει να ανακαλύπτει γρήγορα την λύση του δοσμένου γρίφου.

Κάνοντας μερικά πειράματα για τις δύο οργανώσεις λύσεων παίρνουμε τους παρακάτω πίνακες αποτελεσμάτων. Στην πάνω γραμμή είναι το μέγεθος του πλαισίου και στην κάτω ο χρόνος που πήρε το μοντέλο για να το λύσει.

Πίνακας 33.3: Χρόνος λύσης χωρίς προώθηση των περιορισμών C^l (sec).

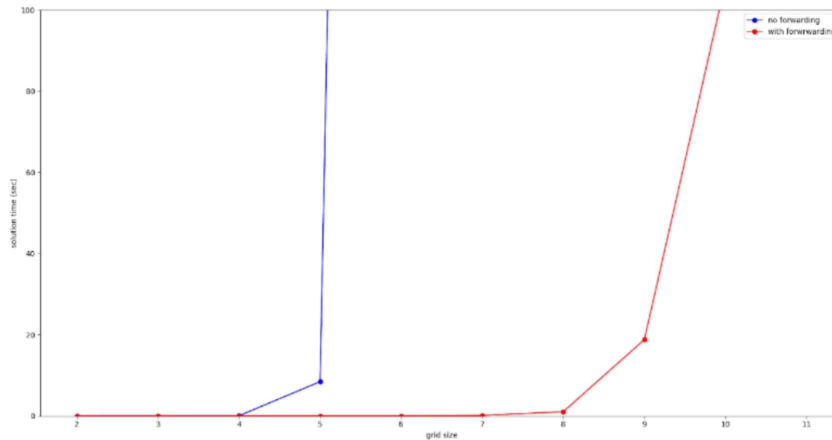
2	3	4	5	6	7	8	9	10	11
0.0015	0.0032	0.0190	8.4175	>30m	>30m	>30m	>30m	>30m	>30m

Πίνακας 33.4: Χρόνος λύσης με προώθηση των περιορισμών C^l (sec).

2	3	4	5	6	7	8	9	10	11
0.0015	0.0012	0.0039	0.0014	0.0032	0.1027	1.0146	18.784	106.65	>30m

Πίνακας 33.5: Χρόνος δημιουργίας λεξικού μεταθέσεων (sec).

2	3	4	5	6	7	8	9	10	11
$<10^{-3}$	$<10^{-3}$	$<10^{-3}$	$<10^{-3}$	$<10^{-3}$	0.0050	0.0528	0.4239	4.3836	51.107



Σχήμα 33.26: Γραφική παράσταση των χρόνων εκτέλεσης.

Μια πρώτη παρατήρηση είναι ότι παρότι το gen_perm_dict παίρνει O(N!) χρόνο να φτιάξει το λεξικό, αυτό δεν επηρεάζει την αναζήτηση καθώς ο χρόνος που παίρνει η αναζήτηση είναι τουλάχιστον δύο τάξεις μεγέθους μεγαλύτερος. Όπου σημειώνεται “>30m” ο αλγόριθμος έτρεξε μέχρι το όριο των 30 λεπτών και δεν βρήκε λύση ακόμα. Όπως φαίνεται η προώθηση που κάνουν αυτόματα τα Or-Tools βελτιώνει σημαντικά την απόδοση της αναζήτησης.

Τέλος με την χρήση του skyscrapers_tester.py γίνονται πειράματα για να εκτιμηθεί η απόδοση των διαφόρων μεθόδων επιλογής επόμενης μεταβλητής-τιμής (πίνακας 3.1). Επιλέγοντας την στρατηγική τιμής SELECT_MIN_VALUE και μέγεθος πλέγματος 9, παρατηρούνται οι παρακάτω μέσοι χρόνοι για την στρατηγική μεταβλητής.

Πίνακας 33.6: Χρόνοι λύσης με διάφορες στρατηγικές επιλογής μεταβλητής.

FIRST	HIGHEST_MAX	LOWEST_MIN	MAX_DOMAIN	MIN_DOMAIN
17.0725	16.0050	16.3367	16.6857	16.5158

Κρατώντας σταθερό το μέγεθος πλέγματος 9 και στρατηγική μεταβλητής την CHOOSE_FIRST μετρήθηκαν οι παρακάτω μέσοι χρόνοι για την στρατηγική τιμής.

Πίνακας 33.7: Χρόνοι λύσης με διάφορες στρατηγικές επιλογής τιμής.

LOWER_HALF	MAX_VALUE	MIN_VALUE	UPPER_HALF
16.7092	16.8447	17.0725	16.8249

Οι χρόνοι που μετρήθηκαν και για τα δύο είδη στρατηγικών έχουν πολύ μικρή μέση απόκλιση σε σχέση με τον μέσο χρόνο εκτέλεσης. Στις στρατηγικές μεταβλητής η μέση απόκλιση είναι 0.1578, με μέσο χρόνο 16.5231. Για τις στρατηγικές τιμής η μέση απόκλιση είναι 0.0231, με μέσο χρόνο 16.8628. Το συμπέρασμα αυτών των μετρήσεων, είναι ότι λόγω της μικρής απόκλισης οι διάφορες στρατηγικές επιλογής επόμενης μεταβλητής-τιμής δεν έχουν κάποια επιρροή στην επίλυση του προβλήματος "Skyscrapers" με την παρούσα μέθοδο.

3.6 Συμπεράσματα

Αυτό το κεφάλαιο αναλύει το παιχνίδι "Skyscrapers" θεωρικά και δίνει μια πρακτική λύση για γρίφους τέτοιου τύπου, μεγέθους πλέγματος μικρότερου ή ίσου με 10. Υλοποιήθηκε ένα πρόγραμμα σε Python το οποίο βρίσκει μια από τις λύσεις ενός δεδομένου γρίφου Skyscrapers. Η υλοποίηση του προγράμματος αυτού έγινε με την βοήθεια του μοντέλου, καθώς αυτό καθοδήγησε την επιλογή τεχνικών CSP για την επιτάχυνση της αναζήτησης.

Κεφάλαιο 4ο: Συμπεράσματα ή/και προτάσεις βελτίωσης

Σε αυτή την εργασία έγινε μια πλήρης ανασκόπηση των τεχνικών CSP. Στην βιβλιογραφία γίνεται σύνδεση μεταξύ των CSP και των τεχνικών Μαθηματικού Προγραμματισμού [7,67]. Η σχέση αυτή και ο Μαθηματικός Προγραμματισμός δεν μελετήθηκαν. Αναλύθηκαν οι μέθοδοι αναζήτησης σε δέντρο που περιλαμβάνουν τεχνικές συνέπειας, τεχνικές διάταξης μεταβλητών και τιμών, στρατηγική οπισθοχώρηση και μεθόδους Look Back και Look Forward για προώθηση περιορισμών. Οι μέθοδοι τοπικής αναζήτησης περιλαμβάνουν τους αλγορίθμους Hill Climbing, Min-Conflicts, Προσομοιούμενη Ανόπτηση, Αναζήτηση Ταμπού και Γενετικούς Αλγορίθμους. Ακόμη παρουσιάζονται οι παραλλαγές των Προβλημάτων Ικανοποίησης Βελτιστοποίησης Περιορισμών και CSP με ελαστικούς περιορισμούς.

Το δεύτερο κεφάλαιο εστιάζει κυρίως στο πρόβλημα του χρονοπρογραμματισμού. Αυτό είναι ένα από τα κυρίαρχα προβλήματα που λύνονται με CSP και έχει πληθώρα από εφαρμογές στην πραγματικότητα.

Το τελευταίο κεφάλαιο δίνει μια πλήρης μοντελοποίηση του παιχνιδιού "Skyscrapers". Το πρόγραμμα που υλοποιήθηκε ίσως δεν είναι το βέλτιστο, γιατί περιορίζεται από το αυστηρό API των OR-Tools. Πιθανώς η επιλογή μιας άλλης βιβλιοθήκης με περισσότερες εκφραστικές δυνατότητες θα ταίριαζε καλύτερα στο Skyscrapers. Ιδιαίτερο ενδιαφέρον θα είχε η μελέτη της απόδοσης ενός αλγορίθμου τοπικής αναζήτησης, όπως ο Min-Conflicts και ο Random Walk (που περιγράφονται στο κεφάλαιο 1.4.1), καθώς οι περιορισμοί C^l (11) για να αποτιμηθούν αποδοτικά απαιτούν να έχουν συμπληρωθεί όλες οι μεταβλητές της γραμμής/στήλης τους.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Βιβλία

[2] B. Porter, V. Lifschitz, and F. Van Harmelen, Eds., Handbook of knowledge representation, 1st ed. Amsterdam ; Boston: Elsevier, 2008.

[19] Selman, Bart, and C.P. Gomes, "Hill-climbing search," Encyclopedia of cognitive science, pp. 333-336, 2006.

[23] F. Glover and M. Laguna, “Tabu Search,” in *Handbook of Combinatorial Optimization*, D.-Z. Du and P. M. Pardalos, Eds. Boston, MA: Springer US, 1998, pp. 2093–2229. doi: 10.1007/978-1-4613-0303-9_33.

[34] S. Boyd, and J. Mattingley, "Branch and bound methods," Notes for EE364b, 2007.

[36] B. G. W. Craenen, A. E. Eiben, and E. Marchiori, “How to handle constraints with evolutionary algorithms,” Practical Handbook Of Genetic Algorithms: Applications, pp. 341-361, 2001.

[38] M. Carver, and L.Conway, "Introduction to VLSI systems," 1980.

Internet Site

[11] R. Barták, “On-line Guide to Constraint Programming,“ 1998, http://kti.mff.cuni.cz/~bartak/constraints/.

[43] P. M. Lellan, “Floorplanning Merged With Synthesis,” 2013, https://semiwiki.com/eda/2813-floorplanning-merged-with-synthesis/.

[53] PuzzleTeamClub, 2022, https://www.puzzle-sudoku.com/

[55] PuzzleTeamClub, 2022, https://www.puzzle-minesweeper.com/

[56] PuzzleTeamClub, 2022, https://www.puzzle-skyscrapers.com/

[59] IBM, 2022, https://www.ibm.com/products/ilog-cplex-optimization-studio

[60] Christian Schulte, 2002, https://www.gecode.org/

[61] Google, 2022, https://developers.google.com/optimization

[68] NumPy, 2022, https://numpy.org/

[69] D. Krupke, “Using and Understanding ortools' CP-SAT: A Primer and Cheat Sheet,” 2022, https://github.com/d-krupke/cpsat-primer

Paper in Conference Proceedings

[3] R. Barták, “Constraint Programming – What is behind?,” Proceedings of the Workshop on Constraint Programming in Decision and Control, Jun. 1999

[4] R. Barták. "Constraint programming: In pursuit of the holy grail." Proceedings of the Week of Doctoral Students (WDS99). vol. 4., 1999.

[7] Jing Gong and Jiaqi Ji, “Integration of Constraint Programming and mathematical programming,” in 2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE), Aug. 2010, pp. V6-451-V6-454. doi: 10.1109/ICACTE.2010.5579785.

[17] I.P. Gent, et al. "The constrainedness of search," AAAI Innovative Applications of Artificial Intelligence Conference, vol. 1,1996.

[18] P.A. Geelen, “Dual viewpoint heuristics for binary constraint satisfaction problems,“ Proceedings of the 10th European Conference on Artificial Intelligence,

pp. 31–35, 1992.

[21] Selman, Bart, and H. Kautz, "Domain-independent extensions to GSAT: Solving large structured satisfiability problems," International Joint Conferences on Artificial Intelligence, Vol. 93, 1993.

[27] C. J. Wang and E. P. K. Tsang, "Solving constraint satisfaction problems using neural networks," 1991 Second International Conference on Artificial Neural Networks, pp. 295-299, 1991.

[30] H. Kanoh, M. Matsumoto, and S. Nishihara, “Genetic algorithms for constraint satisfaction problems,” in 1995 IEEE International Conference on Systems, Man and Cybernetics. Intelligent Systems for the 21st Century, Vancouver, BC, Canada, 1995, vol. 1, pp. 626–631. doi: 10.1109/ICSMC.1995.537833.

[39] H. Eisenmann and F. M. Johannes, “Generic global placement and floorplanning,” in Proceedings of the 35th annual conference on Design automation conference - DAC ’98, San Francisco, California, United States, 1998, pp. 269–274. doi: 10.1145/277044.277119.

[40] Naushad Manzoor Laskar, R. Sen, P. K. Paul, and K. L. Baishnab, “A survey on VLSI Floorplanning: Its representation and modern approaches of optimization,” in 2015 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS), Coimbatore, India, Mar. 2015, pp. 1–9. doi: 10.1109/ICIIECS.2015.7192989.

[41] D. F. Wong and C. L. Liu, “A New Algorithm for Floorplan Design,” in 23rd ACM/IEEE Design Automation Conference, Las Vegas, Nevada, USA, 1986, pp. 101–107. doi: 10.1109/DAC.1986.1586075.

[46] H. A. Razak, Z. Ibrahim, and N. M. Hussin, “Bipartite graph edge coloring approach to course timetabling,” in 2010 International Conference on Information Retrieval & Knowledge Management (CAMP), Shah Alam, Selangor, Malaysia, Mar. 2010, pp. 229–234. doi: 10.1109/INFRKM.2010.5466912.

[47] A. Dandashi and M. Al-Mouhamed, “Graph Coloring for class scheduling,” in ACS/IEEE International Conference on Computer Systems and Applications - AICCSA 2010, Hammamet, Tunisia, May 2010, pp. 1–4. doi: 10.1109/AICCSA.2010.5586963.

[48] Lixi Zhang and SimKim Lau, “Constructing university timetable using constraint satisfaction programming approach,” in International Conference on Computational Intelligence for Modelling,

Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06), Vienna, Austria, 2005, vol. 2, pp. 55–60. doi: 10.1109/CIMCA.2005.1631445.

[51] P. Khonggamnerd and S. Innet, "On Improvement of Effectiveness in Automatic University Timetabling Arrangement with Applied Genetic Algorithm," in 2009 Fourth International Conference on Computer Sciences and Convergence Information Technology, Seoul, Korea, 2009, pp. 1266–1270. doi: 10.1109/ICCIT.2009.202.

[52] O. M. K. Alsmadi, Z. S. Abo-Hammour, D. I. Abu-Al-Nadi, and A. Algsoon, "A novel genetic algorithm technique for solving university course timetabling problems," in International Workshop on Systems, Signal Processing and their Applications, WOSSPA, Tipaza, Algeria, May 2011, pp. 195–198. doi: 10.1109/WOSSPA.2011.5931449.

[57] M. Tuga, R. Berretta, and A. Mendes, "A Hybrid Simulated Annealing with Kempe Chain Neighborhood for the University Timetabling Problem," in 6th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2007), Melbourne, Australia, 2007, pp. 400–405. doi: 10.1109/ICIS.2007.25.

[64] F. Galea, and B. L. Cun, "Bob++: a framework for exact combinatorial optimization methods on parallel machines," International Conference High Performance Computing & Simulation, 2007.

[65] T. Menouer and B. L. Cun, "A Parallelization Mixing OR-Tools/Gecode Solvers on Top of the Bobpp Framework," in 2013 Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, COMPIEGNE, France, Oct. 2013, pp. 242–246. doi: 10.1109/3PGCIC.2013.42.

[66] C. Schulte, "Oz Explorer: A visual constraint programming tool," International Symposium on Programming Language Implementation and Logic Programming, 1996.

Journal Articles

[1] H. Babaei, J. Karimpour, and A. Hadidi, "A survey of approaches for university course timetabling problem," Computers & Industrial Engineering, vol. 86, pp. 43–59, Aug. 2015, doi: 10.1016/j.cie.2014.11.010.

[5] P. Dabas, and V. Cooner, "A Review of Constraint Programming," *International Journal of Computer Applications Technology and Research*, vol. 3, no. 7, pp. 395 - 399, 2014, ISSN: 2319–8656

[6] P. Van Hentenryck, L. Michel, L. Perron, and J.-C. Régin, "Constraint Programming in OPL," in Principles and Practice of Declarative Programming, vol. 1702, 1999, pp. 98–116. doi: 10.1007/10704567_6.

[8] J. Rodriguez, "A constraint programming model for real-time train scheduling at junctions," Transportation Research Part B: Methodological, vol. 41, no. 2, pp. 231–245, Feb. 2007, doi: 10.1016/j.trb.2006.02.006.

[9] E. C. Freuder, "Synthesizing constraint expressions," Commun. ACM, vol. 21, no. 11, pp. 958–966, Nov. 1978, doi: 10.1145/359642.359654.

[10] U. Montanari, “Networks of constraints: Fundamental properties and applications to picture processing,” Information Sciences, vol. 7, pp. 95–132, Jan. 1974, doi: 10.1016/0020-0255(74)90008-5.

[12] D. Waltz, "Understanding line drawings of scenes with shadows," Psychology of Computer Vision, pp. 19-91, 1975.

[13] R. M. Haralick and G. L. Elliott, “Increasing tree search efficiency for constraint satisfaction problems,” *Artificial Intelligence*, vol. 14, no. 3, pp. 263–313, Oct. 1980, doi: 10.1016/0004-3702(80)90051-X.

[14] C. Bessière and J.-C. Régin, “MAC and combined heuristics: Two reasons to forsake FC (and CBJ?) on hard problems,” *Principles and Practice of Constraint Programming — CP96*, vol. 1118, pp. 61–75, 1996. doi: 10.1007/3-540-61551-2_66.

[15] D. Brélaz, “New methods to color the vertices of a graph,” *Commun. ACM*, vol. 22, no. 4, pp. 251–256, Apr. 1979, doi: 10.1145/359094.359101.

[16] I. P. Gent, E. MacIntyre, P. Presser, B. M. Smith, and T. Walsh, “An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem,” *Principles and Practice of Constraint Programming — CP96*, vol. 1118, 1996, pp. 179–193. doi: 10.1007/3-540-61551-2_74.

[20] S. Minton, M. D. Johnston, A. B. Philips, and P. Laird, “Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems,” *Artificial Intelligence*, vol. 58, no. 1–3, pp. 161–205, Dec. 1992, doi: 10.1016/0004-3702(92)90007-K.

[22] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by Simulated Annealing,” Science, vol. 220, no. 4598, pp. 671–680, May 1983, doi: 10.1126/science.220.4598.671.

[24] A. K. Mackworth, “Consistency in networks of relations,” Artificial Intelligence, vol. 8, no. 1, pp. 99–118, Feb. 1977, doi: 10.1016/0004-3702(77)90007-8.

[25] I. Rivin, I. Vardi, and P. Zimmermann, “The *n* -Queens Problem,” *The American Mathematical Monthly*, vol. 101, no. 7, pp. 629–639, Aug. 1994, doi: 10.1080/00029890.1994.11997004.

[26] S. C. Brailsford, C. N. Potts, and B. M. Smith, “Constraint satisfaction problems: Algorithms and applications,” European Journal of Operational Research, vol. 119, no. 3, pp. 557–581, Dec. 1999, doi: 10.1016/S0377-2217(98)00364-6.

[28] M. W. Gardner and S. R. Dorling, “Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences,” Atmospheric Environment, vol. 32, no. 14–15, pp. 2627–2636, Aug. 1998, doi: 10.1016/S1352-2310(97)00447-0.

[29] S. Katoch, S. S. Chauhan, and V. Kumar, “A review on genetic algorithm: past, present, and future,” Multimed Tools Appl, vol. 80, no. 5, pp. 8091–8126, Feb. 2021, doi: 10.1007/s11042-020-10139-6.

[31] H. Kanoh, M. Matsumoto, K. Hasegawa, N. Kato, and S. Nishihara, “Solving constraint-satisfaction problems by a genetic algorithm adopting viral infection,” Engineering Applications of Artificial Intelligence, vol. 10, no. 6, pp. 531–537, Dec. 1997, doi: 10.1016/S0952-1976(97)00035-3.

[32] R. Tarjan, “Depth-First Search and Linear Graph Algorithms,” SIAM J. Comput., vol. 1, no. 2, pp. 146–160, Jun. 1972, doi: 10.1137/0201010.

[33] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis, “How easy is local search?,” Journal of Computer and System Sciences, vol. 37, no. 1, pp. 79–100, Aug. 1988, doi: 10.1016/0022-0000(88)90046-3.

[35] E. C. Freuder and R. J. Wallace, “Partial constraint satisfaction,” Artificial Intelligence, vol. 58, no. 1–3, pp. 21–70, Dec. 1992, doi: 10.1016/0004-3702(92)90004-H.

[37] B. G. W. Craenen, A. E. Eiben, and J. I. van Hemert, “Comparing evolutionary algorithms on binary constraint satisfaction problems,” IEEE Trans. Evol. Computat., vol. 7, no. 5, pp. 424–444, Oct. 2003, doi: 10.1109/TEVC.2003.816584

[42] M. Rebaudengo and M. S. Reorda, “GALLO: a genetic algorithm for floorplan area optimization,” IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 15, no. 8, pp. 943–951, Aug. 1996, doi: 10.1109/43.511573.

[44] A. Borning, B. Freeman-Benson, and M. Wilson, “Constraint hierarchies,” Lisp and Symbolic Computation, vol. 5, no. 3, pp. 223–270, Sep. 1992, doi: 10.1007/BF01807506.

[45] D. J. A. Welsh, “An upper bound for the chromatic number of a graph and its application to timetabling problems,” The Computer Journal, vol. 10, no. 1, pp. 85–86, Jan. 1967, doi: 10.1093/comjnl/10.1.85.

[49] S. Deris, S. Omatu, and H. Ohta, “Timetable planning using the constraint-based reasoning,” Computers & Operations Research, vol. 27, no. 9, pp. 819–840, Aug. 2000, doi: 10.1016/S0305-0548(99)00051-9.

[50] S. Deris, S. Omatu, H. Ohta, and P. Saad, “Incorporating constraint propagation in genetic algorithm for university timetable planning,” Engineering Applications of Artificial Intelligence, vol. 12, no. 3, pp. 241–253, Jun. 1999, doi: 10.1016/S0952-1976(99)00007-X.

[54] T. Rokicki, et al, "The Diameter of the Rubik's Cube Group Is Twenty," SIAM Journal on Discrete Mathematics, 2013.

[58] C. H. Aladag, G. Hocaoglu, M. A. Basaran, “The effect of neighborhood structures on tabu search algorithm in solving course timetabling problem,” in Expert Systems with Applications,

Vol. 36, 2009, pp. 12349-12356.

[62] T. Schrijvers, P. Stuckey, and P. Wadler, “Monadic constraint programming,” J. Funct. Prog., vol. 19, no. 6, pp. 663–697, Nov. 2009, doi: 10.1017/S0956796809990086.

[63] P. Hofstedt, “Constraint-Based Object-Oriented Programming,” IEEE Softw., vol. 27, no. 5, pp. 53–56, Sep. 2010, doi: 10.1109/MS.2010.89.

[67] I. J. Lustig and J.-F. Puget, “Program Does Not Equal Program: Constraint Programming and Its Relationship to Mathematical Programming,” Interfaces, vol. 31, no. 6, pp. 29–53, Dec. 2001, doi: 10.1287/inte.31.6.29.9647.

ΠΑΡΆΡΤΗΜΑ Α: Κώδικας skyscrapers_solver.py

```
import time
import numpy as np
from itertools import permutations
from ortools.sat.python import cp_model

class SolutionPrinter(cp_model.CpSolverSolutionCallback):

    def __init__(self, vars, side_lists):
        cp_model.CpSolverSolutionCallback.__init__(self)        # init super class
        self.__solution_count = 0                               # internal solution count
        self.__V = vars                                         # variable handles
        self.__side_lists = side_lists                          # side lists
        self.__size = len(vars)                                 # internal grid size

    def solution_count(self):                                   # get solution count
        return self.__solution_count

    def on_solution_callback(self):                             # solution callback

        self.__solution_count += 1

        print('\n'+(self.__size+3)*2*'-')                       # top separating lines

        print(end='   ')                                        # print top side list
        for j in range(self.__size):
            print(self.__side_lists[0,j],end=' ')
        print('\n   '+self.__size*'V ')

        for i in range(self.__size):                            # print skyscaper grid
            print(str(self.__side_lists[1,i]) + '>', end=' ')   # print left side list
            for j in range(self.__size):
                print(self.Value(self.__V[i,j]), end=' ')
            print('<' + str(self.__side_lists[2,i]), end=' ')   # print right side list
            print()

        print('   '+self.__size*'Λ ')                           # print bottom side list
        print(end='   ')
        for j in range(self.__size):
            print(self.__side_lists[3,j],end=' ')
        print()

        print((self.__size+3)*2*'-')                            # bottom separating lines

        self.StopSearch()                                       # stop searching


def visible(row, max_val):
    seen = 1                                                    # seen skyscrapers
    cur_max = row[0]                                            # current max

    for scr in row[1:]:                                          # iterate through row from 2nd
element
        if scr > cur_max:                                        # if this skyscraper is taller
than the previous maximum
            seen += 1                                           # it can be seen
            cur_max = scr                                       # and is now the new maximum
            if cur_max == max_val:                                  # can't see past the gobal
maximum
                break
    return seen


def gen_perm_dict(grid_size):
    vals = [x for x in range(1, grid_size+1)]                   # all valid values
    perms = permutations(vals)                                      # all permutations of the
values
    dict = {}                                                           # dictionary of all
permutations
    for perm in perms:
```

```
        left = visible(perm, grid_size)                         # visible from left
        right = visible(perm[::-1], grid_size)                  # visible from right
        key = (left, right)                                     # key for this permutation
        if  key not in dict.keys():                             # add this perm to dictionary
            dict[key] = [perm]
        else:
            dict[key].append(perm)

    return dict


def set_variables(grid_size, model, domain):
    variables = []                                                      # temporary 2D array of
variables
    for i in range(grid_size):                                  # fill the variable array
        variables.append([])
        for j in range(grid_size):
            var = model.NewIntVar(domain[0], domain[1], 'v'+str(i)+','+str(j))
            variables[-1].append(var)

    return np.array(variables)                                  # numpy array for easy slicing


def set_constraints(grid_size, model, preplaced, top, right, bottom, left, assignments):
    for i in range(grid_size):                                      #add all rows and coloumns
have unique numbers constraint
        model.AddAllDifferent(V[i,:].tolist())
        model.AddAllDifferent(V[:,i].tolist())

    for pre in preplaced:                                               # add preplaced values
constraint to model
        model.AddHint(V[pre[0],pre[1]], pre[2])

    for idx,key in enumerate(zip(left, right)):                 # add allowed assignments, key
is made from zipping (l,r)
        if key[0]>0 and key[1]>0:                               # key exists
            model.AddAllowedAssignments(V[idx,:].tolist(), assignments[key])


        elif key[0]>0 and key[1]==0:                            # only left part of key exists
            combined_assignments = []                           # combine all assignments with
this left part
            for i in range(1, grid_size+1):
                k = (key[0],i)                                  # new key
                if k in assignments.keys():                            # if it exists add it's
assignments to the total list
                    combined_assignments += assignments[k]
            model.AddAllowedAssignments(V[idx,:].tolist(), combined_assignments)

        elif key[0]==0 and key[1]>0:                                 # only right part of key
exists
            combined_assignments = []                           # combine all assignments with
this right part
            for i in range(1, grid_size+1):
                k = (i,key[1])                                  # new key
                if k in assignments.keys():                            # if it exists add it's
assignments to the total list
                    combined_assignments += assignments[k]
            model.AddAllowedAssignments(V[idx,:].tolist(), combined_assignments)



    for idx,key in enumerate(zip(top, bottom)):                 # add allowed assignments, key
is made from zipping (t,b)
        if key[0]>0 and key[1]>0:                               # key exists
            model.AddAllowedAssignments(V[:,idx].tolist(), assignments[key])


        elif key[0]>0 and key[1]==0:                            # only top part of key exists
            combined_assignments = []                           # combine all assignments with
this top part
            for i in range(1, grid_size+1):
                k = (key[0],i)                                  # new key
                if k in assignments.keys():                            # if it exists add it's
assignments to the total list
                    combined_assignments += assignments[k]
            model.AddAllowedAssignments(V[:,idx].tolist(), combined_assignments)
```

```
        elif key[0]==0 and key[1]>0:                                # only bottom part of key
exists
            combined_assignments = []                               # combine all assignments with
this bottom part
            for i in range(1, grid_size+1):
                k = (i,key[1])                                      # new key
                if k in assignments.keys():                               # if it exists add it's
assignments to the total list
                    combined_assignments += assignments[k]
            model.AddAllowedAssignments(V[:,idx].tolist(), combined_assignments)


def print_status(status):
    if status == cp_model.OPTIMAL:
        print('OPTIMAL,', solution_printer.solution_count(),'solutions')
    elif status == cp_model.FEASIBLE:
        print('FEASIBLE,', solution_printer.solution_count(),'solutions')
    elif status == cp_model.INFEASIBLE:
        print('INFEASIBLE')
    elif status == cp_model.MODEL_INVALID:
        print('MODEL_INVALID')
    elif status == cp_model.UNKNOWN:
        print('UNKNOWN')


############################## MAIN ############################

grid_size = 4                                                       # NxN grid size

domain = (1, grid_size)                                             # domain for all the variables

preplaced = []                                                      # preplaced values constraint

top     = [1,2,3,2]                                                 # sides constraints
right   = [2,3,1,2]                                                    # top/bottom: left -> right
index
bottom  = [3,2,1,2]                                                 # right/left: up -> down index
left    = [1,2,2,3]

model = cp_model.CpModel()                                          # make model object

V = set_variables(grid_size, model, domain)                           # generate the variables and
add them to the model

assignments = gen_perm_dict(grid_size)                              # generate allowed assignments
dictionary

set_constraints(grid_size, model, preplaced,                        # generate the constraints and
add them to the model
                top, right, bottom, left, assignments)

solver = cp_model.CpSolver()                                        # make solver object

solver.parameters.enumerate_all_solutions = True                    # required for StopSearch() in
solution_printer to work
side_lists = np.array([top, left, right, bottom])                   # for solution_printer
solution_printer = SolutionPrinter(V, side_lists)                   # make printer object

start = time.time()
status = solver.Solve(model, solution_printer)                      # solve the problem
print("time:", time.time() - start)

print_status(status)                                                # print status
```

ΠΑΡΆΡΤΗΜΑ Β: Κώδικας skyscrapers_tester.py

```
import time
import numpy as np
from itertools import permutations
from ortools.sat.python import cp_model

class SolutionPrinter(cp_model.CpSolverSolutionCallback):

    def __init__(self, vars, side_lists, forward):
        cp_model.CpSolverSolutionCallback.__init__(self)        # init super class
        self.__solution_count = 0                               # internal solution count
        self.__V = vars                                         # variable handles
        self.__side_lists = side_lists                          # side lists
        self.__size = len(vars)                                 # internal grid size
        self.__forw = forward                                    # whether this model forwards
constraints

    def solution_count(self):                                   # get solution count
        return self.__solution_count

    def on_solution_callback(self):                             # solution callback
        self.__solution_count += 1

        if not forward:
            for i in range(self.__size):                         # checks if solution is valid,
returns if not to continue search
                row = [self.Value(x) for x in self.__V[i,:]]
                col = [self.Value(x) for x in self.__V[:,i]]
                if(visible(col,self.__size) != self.__side_lists[0][i]):        # compare with
top
                    return
                if(visible(row[::-1],self.__size) != self.__side_lists[1][i]):  # compare with
right
                    return
                if(visible(col[::-1],self.__size) != self.__side_lists[2][i]):  # compare with
bottom
                    return
                if(visible(row,self.__size) != self.__side_lists[3][i]):        # compare with
left
                    return
        self.StopSearch()                                       # stop searching if valid


def visible(row, max_val):
    seen = 1                                                    # seen skyscrapers
    cur_max = row[0]                                            # current max

    for scr in row[1:]:                                          # iterate through row from 2nd
element
        if scr > cur_max:                                        # if this skyscraper is taller
than the previous maximum
            seen += 1                                           # it can be seen
            cur_max = scr                                       # and is now the new maximum
            if cur_max == max_val:                                  # can't see past the gobal
maximum
                break
    return seen

def gen_perm_dict(grid_size):
    vals = [x for x in range(1, grid_size+1)]                   # all valid values
    perms = permutations(vals)                                       # all permutations of the
values
    dict = {}                                                         # dictionary of all
permutations
    for perm in perms:
        left = visible(perm, grid_size)                         # visible from left
        right = visible(perm[::-1], grid_size)                  # visible from right
        key = (left, right)                                     # key for this permutation
        if  key not in dict.keys():                             # add this perm to dictionary
            dict[key] = [perm]
        else:
            dict[key].append(perm)
```

```
    return dict

def set_variables(grid_size, model, domain):
    variables = []                                                      # temporary 2D array of
variables
    for i in range(grid_size):                                       # fill the variable array
        variables.append([])
        for j in range(grid_size):
            var = model.NewIntVar(domain[0], domain[1], 'v'+str(i)+','+str(j))
            variables[-1].append(var)

    return np.array(variables)                                          # numpy array for easy
slicing

def set_constraints(grid_size, model, V, preplaced, top, right, bottom, left, assignments,
forward):
    for i in range(grid_size):                                      #add all rows and coloumns
have unique numbers constraint
        model.AddAllDifferent(V[i,:].tolist())
        model.AddAllDifferent(V[:,i].tolist())

    for pre in preplaced:                                               # add preplaced values
constraint to model
        model.AddHint(V[pre[0],pre[1]], pre[2])


    if forward:
        for idx,key in enumerate(zip(left, right)):                 # add allowed assignments,
key is made from zipping (l,r)
            if key[0]>0 and key[1]>0:                                # key exists
                model.AddAllowedAssignments(V[idx,:].tolist(), assignments[key])


            elif key[0]>0 and key[1]==0:                                 # only left part of key
exists
                combined_assignments = []                              # combine all assignments
with this left part
                for i in range(1, grid_size+1):
                    k = (key[0],i)                                    # new key
                    if k in assignments.keys():                          # if it exists add it's
assignments to the total list
                        combined_assignments += assignments[k]
                model.AddAllowedAssignments(V[idx,:].tolist(), combined_assignments)


            elif key[0]==0 and key[1]>0:                                # only right part of key
exists
                combined_assignments = []                              # combine all assignments
with this right part
                for i in range(1, grid_size+1):
                    k = (i,key[1])                                    # new key
                    if k in assignments.keys():                          # if it exists add it's
assignments to the total list
                        combined_assignments += assignments[k]
                model.AddAllowedAssignments(V[idx,:].tolist(), combined_assignments)



        for idx,key in enumerate(zip(top, bottom)):                 # add allowed assignments,
key is made from zipping (t,b)
            if key[0]>0 and key[1]>0:                                # key exists
                model.AddAllowedAssignments(V[:,idx].tolist(), assignments[key])


            elif key[0]>0 and key[1]==0:                                  # only top part of key
exists
                combined_assignments = []                              # combine all assignments
with this top part
                for i in range(1, grid_size+1):
                    k = (key[0],i)                                    # new key
                    if k in assignments.keys():                          # if it exists add it's
assignments to the total list
                        combined_assignments += assignments[k]
                model.AddAllowedAssignments(V[:,idx].tolist(), combined_assignments)


            elif key[0]==0 and key[1]>0:                               # only bottom part of key
exists
```

```
                combined_assignments = []                            # combine all assignments
with this bottom part
                for i in range(1, grid_size+1):
                    k = (i,key[1])                                   # new key
                    if k in assignments.keys():                         # if it exists add it's
assignments to the total list
                        combined_assignments += assignments[k]
                model.AddAllowedAssignments(V[:,idx].tolist(), combined_assignments)

def print_status(status):
    if status == cp_model.OPTIMAL:
        print('OPTIMAL,', solution_printer.solution_count(),'solutions tries')
    elif status == cp_model.FEASIBLE:
        print('FEASIBLE,', solution_printer.solution_count(),'solutions tries')
    elif status == cp_model.INFEASIBLE:
        print('INFEASIBLE')
    elif status == cp_model.MODEL_INVALID:
        print('MODEL_INVALID')
    elif status == cp_model.UNKNOWN:
        print('UNKNOWN')

############################## MAIN ############################

forward = True                                               # use forwarding
R = range(8,9)                                                # set the range from smallest
grid size to largest(+1)

preplaced = []                                               # preplaced values constraint

L2 =  [2,1, 1,2, 2,1, 1,2]                                   # generated games
L3 =  [2,2,1, 1,2,2, 3,1,2, 2,1,3]
L4 =  [1,2,3,2, 2,3,1,2, 2,1,2,3, 3,2,2,1]
L5 =  [2,3,1,4,3, 2,3,2,1,2, 2,1,4,2,2, 3,5,2,1,2]
L6 =  [2,4,3,2,1,3, 2,2,4,4,5,1, 1,2,5,3,2,3, 4,2,3,1,4,2]
L7 =  [4,3,3,3,1,2,2, 2,2,3,3,5,3,1, 1,3,6,4,4,2,2, 2,2,1,3,3,3,4]
L8 =  [4,2,3,1,3,2,4,2, 2,4,4,2,2,1,5,2, 3,1,2,4,3,3,3,2, 5,1,4,2,3,5,2,3]
L9 =  [2,1,3,4,4,3,3,2,4, 3,8,4,4,2,2,3,1,2, 2,1,2,4,3,2,3,8,3, 4,2,2,2,4,3,3,1,2]
L10 = [7,4,5,5,2,2,2,2,1,4, 2,3,2,3,1,5,4,5,6,7, 6,6,5,4,5,4,1,3,2,2, 3,1,2,2,2,4,3,4,3,7]
L11     =     [1,2,3,3,3,5,4,3,3,2,4,     3,9,5,4,3,1,2,2,3,4,3,     5,2,3,1,2,3,2,4,5,9,3,
5,2,2,3,4,5,3,3,3,2,1]

D = {2:L2,3:L3,4:L4,5:L5,6:L6,7:L7,8:L8,9:L9,10:L10}           # generated games dict

for j in R:
    print('\n\n',j,':')

    grid_size = j                                            # set parameters
    domain = (1, grid_size)
    L = D[j]

    assignments = gen_perm_dict(grid_size)                    # generate allowed assignments
dictionary

    for i in range(4):                                       # set sides constraints
        top =       L[0:grid_size]
        right =     L[grid_size:2*grid_size]
        bottom =    L[2*grid_size:3*grid_size][::-1]
        left =      L[3*grid_size:4*grid_size][::-1]

        temp = []                                                # left shift the generated
game list by grid_size
        for j in range(grid_size):
            temp.append(L.pop(0))
        L+=temp

        model = cp_model.CpModel()                           # make model object

        V = set_variables(grid_size, model, domain)              # generate the variables and
constraints and add them to the model

        model.AddDecisionStrategy(list(V.reshape((grid_size*grid_size,))),
cp_model.CHOOSE_FIRST, cp_model.SELECT_MIN_VALUE)
 # CHOOSE_FIRST             # SELECT_LOWER_HALF
 # CHOOSE_HIGHEST_MAX       # SELECT_MAX_VALUE
 # CHOOSE_LOWEST_MIN        # SELECT_MIN_VALUE
```

```
 # CHOOSE_MAX_DOMAIN_SIZE    # SELECT_UPPER_HALF
 # CHOOSE_MIN_DOMAIN_SIZE

        set_constraints(grid_size, model, V, preplaced, top, right, bottom, left, assignments,
forward)

        solver = cp_model.CpSolver()                            # make solver object

        solver.parameters.enumerate_all_solutions = True                       # required for
StopSearch() in solution_printer to work
        side_lists = np.array([top, right, bottom, left])               # for solution_printer
        solution_printer = SolutionPrinter(V, side_lists, forward)      # make printer object

        start = time.time()
        status = solver.Solve(model, solution_printer)                  # solve the problem
        print("time:", time.time() - start)

        print_status(status)                                            # print status
```