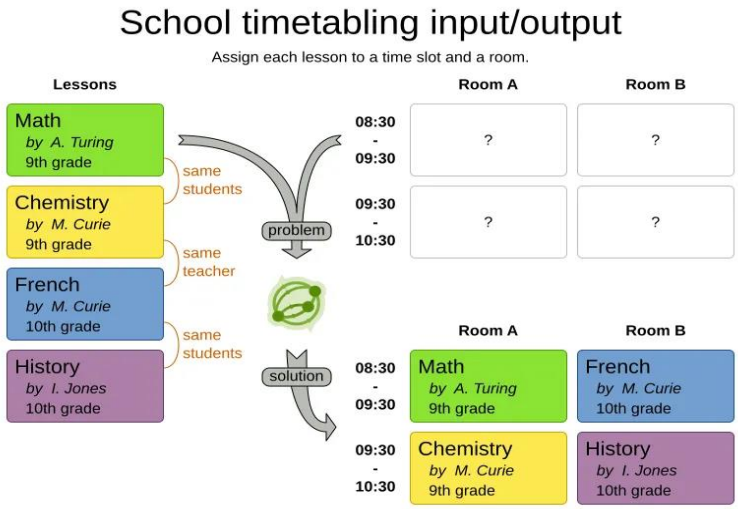
ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«ΚΑΤΑΡΤΙΣΗ ΩΡΟΛΟΓΙΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ ΜΕ ΤΕΧΝΙΚΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΙΚΑΝΟΠΟΙΗΣΗΣ ΠΕΡΙΟΡΙΣΜΩΝ»



Του φοιτητή
Μότσια Γεώργιου
Αρ. Μητρώου: 154498

Επιβλέπων
Ευστάθιος Αντωνίου
Καθηγητής

Ημερομηνία 16/09/2023

Τίτλος Π.Ε.: Κατάρτιση ωρολογίου προγράμματος με τεχνικές προγραμματισμού ικανοποίησης περιορισμών

Κωδικός Π.Ε.: 21228

Ονοματεπώνυμο φοιτητή: Μότσιας Γεώργιος

Ονοματεπώνυμο εισηγητή: Αντωνίου Ευστάθιος

Ημερομηνία ανάληψης Π.Ε.: 31-03-2021

Ημερομηνία περάτωσης Π.Ε.: 16/09/2023

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 έγκριση της διπλωματικής 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.

Πρόλογος

Επέλεξα αυτό το θέμα για την Πτυχιακή Εργασία μου επειδή ενδιαφερόμουν ιδιαίτερα για τον τομέα του προγραμματισμού ικανοποίησης περιορισμών και την εφαρμογή του στον χώρο της κατάρτισης ωρολογίου προγράμματος. Τα οφέλη περιλάμβαναν τη βελτίωση της αποδοτικότητας στον προγραμματισμό ωρολογίου προγράμματος, τη μείωση των συγκρούσεων και τη δυνατότητα εύκολης προσαρμογής σε αλλαγές. Επιπλέον, αυτό το θέμα επέτρεψε την εφαρμογή προηγμένων αλγορίθμων προγραμματισμού, ενισχύοντας τις γνώσεις μου στον τομέα της πληροφορικής.

Περίληψη

Στην παρούσα Πτυχιακή Εργασία αναπτύχθηκε μια εφαρμογή σε προγραμματιστική γλώσσα Python με χρήση της βιβλιοθήκης Google OR-Tools που έχει ως στόχο την δημιουργία ωρολογίου προγράμματος σε ιδρύματα Πρωτοβάθμιας και Δευτεροβάθμιας Εκπαίδευσης στον ελλαδικό χώρο. Στο πρώτο κεφάλαιο γίνεται η εισαγωγή στο πρόβλημα, όπου παρουσιάζονται γενικά μελέτες που προσπάθησαν να αντιμετωπίσουν το ίδιο πρόβλημα, ορίζονται οι περιορισμοί που θα τεθούν προς επίλυση και αναλύονται οι ορολογίες που θα χρησιμοποιηθούν. Στο δεύτερο κεφάλαιο αναλύεται ο όρος Προβλήματα Ικανοποίησης Περιορισμών, δίνοντας κάποια ενδελεχή παραδείγματα και στην συνέχεια αναλύεται πως ο προγραμματισμός επιχειρεί να τα αντιμετωπίσει και να τα επιλύσει. Στο τρίτο κεφάλαιο παρουσιάζεται η βιβλιοθήκη OR-Tools και ποια μέρη της χρησιμοποιήθηκαν για την υλοποίηση της εφαρμογής. Στο τέταρτο κεφάλαιο δημιουργείται το μαθηματικό μοντέλο του προβλήματος και έπειτα αναλύεται η εφαρμογή καθώς και ο τρόπος χρήσης της. Στο πέμπτο κεφάλαιο παρουσιάζονται τα αποτελέσματα, τα οποία κάνουν σαφές ότι μια εφαρμογή τέτοιου τύπου δεν μπορεί να αντικαταστήσει πλήρως την ανθρώπινη παρουσία, όμως μπορεί να παρέχει μεγάλη ταχύτητα δίνοντας ένα πολύ επαρκές πλάνο για την υλοποίηση του ωρολογίου προγράμματος.

«ESTABLISHMENT OF A TIMETABLE WITH CONSTRAINT SATISFACTION PROGRAMMING TECHNIQUES»

«George Motsias»

Abstract

In this thesis an application in Python programming language was developed using the Google OR-Tools library that aims to create a timetable in Primary and Secondary Education institutions in Greece. The first chapter introduces the problem, presenting general studies that have tried to solve the same problem, defining the constraints that will be set to be solved and analyzing the terminologies that will be used. The second chapter discusses the term Constraint Satisfaction Problems, giving some thorough examples, and then discusses how programming attempts to address and solve them. The third chapter presents the OR-Tools library and which parts of it were used to implement the application. In chapter four, the mathematical model of the problem is created and then the application and how to use it are discussed. In the fifth chapter the results are presented, which make it clear that an application of this type cannot completely replace human presence, but it can provide great speed by giving a very adequate plan for the implementation of the timetable.

Περιεχόμενα

Κατάλογος Σχημάτων

Κεφάλαιο 1ο: Εισαγωγή στο πρόβλημα

1.1 Εισαγωγή

Η παρούσα Πτυχιακή Εργασία εντάσσεται στο πλαίσιο της Σχολής Μηχανικών του τμήματος Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων. Σαν στόχο έχει την υλοποίηση ενός προγράμματος σε προγραμματιστική γλώσσα Python με χρήση του λογισμικού OR-Tools το οποίο θα είναι σε θέση να δημιουργεί ένα λειτουργικό ωρολόγιο σχολικό πρόγραμμα Πρωτοβάθμιας ή Δευτεροβάθμιας εκπαίδευσης. Σκοπός είναι να αναδείξει την δυσκολία που αντιμετωπίζουν τα εκπαιδευτικά ιδρύματα κάθε χρόνο για την δημιουργία ενός λειτουργικού ωρολογίου προγράμματος, το οποίο θα ικανοποιεί ένα μεγάλο εύρος περιορισμών και πως μπορεί η τεχνολογία και συγκεκριμένα οι αυτοματισμοί των αλγορίθμων να μας διευκολύνουν στη αποπεράτωσή του. Μπορεί το αποτέλεσμα να μην είναι τέλειο, αλλά μας δίνει σίγουρα ένα αρχικό στήριγμα και πολλές φορές με μικρές προσαρμογές μπορούμε να καταλήξουμε σε ένα ολοκληρωμένο ωρολόγιο πρόγραμμα.

Στο Κεφάλαιο 1 θα γίνει μια βιβλιογραφική ανασκόπηση του προβλήματος, θα αναλυθεί το πρόβλημα που προσπαθήσουμε να λύσουμε και θα δοθούν οι περιορισμοί. Στο Κεφάλαιο 2 θα εξηγήσουμε τι είναι τα προβλήματα περιορισμών και στη συνέχεια θα αναλυθούν κάποια από τα πιο γνωστά προβλήματα περιορισμών που υπάρχουν και ο τρόπος με τον οποίο επιλύονται. Στο Κεφάλαιο 3 θα αναλύσουμε το λογισμικό Google OR-Tools, πως λειτουργεί και ποιες είναι οι συναρτήσεις που θα χρησιμοποιηθούν. Στο Κεφάλαιο 4 θα γίνει η ανάλυση του προβλήματος, θα παρουσιάσουμε το μαθηματικό μοντέλο και θα αναλύσουμε το λογισμικό σε επίπεδο προγραμματισμού και σε επίπεδο χρήσης. Τέλος στο Κεφάλαιο 5 θα παρουσιάσουμε τα συμπεράσματα της παρούσας Πτυχιακής Εργασίας.

1.2 Το πρόβλημα

Τα προβλήματα χρονοπρογραμματισμού εκδηλώνονται σε διάφορους τομείς της πρακτικής και της έρευνας και μπορούν να περιγραφούν ως το έργο της κατανομής των πόρων στις διαθέσιμες χρονοθυρίδες, έτσι ώστε ένα σύνολο περιορισμών να ικανοποιούνται. Επιπλέον, ένα σύνολο ποιοτικών χαρακτηριστικών καθιστά τα εναλλακτικά ωρολόγια προγράμματα ανώτερα ή κατώτερα μεταξύ τους. Τα προβλήματα χρονοπρογραμματισμού στη γενική τους μορφή ανήκουν στην κατηγορία των NP-complete[1] προβλημάτων που δίνουν λίγες ελπίδες για την εύρεση αλγορίθμου που παράγει μια βέλτιστη λύση σε πολυωνυμικό περιορισμένο χρόνο. Ωστόσο, συγκεκριμένα προβλήματα χρονοπρογραμματισμού με μεγάλο πρακτικό ενδιαφέρον μπορούν να επιλυθούν ικανοποιητικά. Ως εκ τούτου, έχει αναπτυχθεί ένας πλούτος προσεγγίσεων επίλυσης που προέρχονται κυρίως από τον Μαθηματικό Προγραμματισμό, την Υπολογιστική Νοημοσύνη και την Μεταευρετική.

Τα εκπαιδευτικά προβλήματα χρονοπρογραμματισμού αποτελούν ειδική περίπτωση των προβλημάτων χρονοπρογραμματισμού. Έχουν μελετηθεί λεπτομερώς και, ακόμη και πριν από τη νέα χιλιετία, έχει αναπτυχθεί πληθώρα προσεγγίσεων όπως μπορεί να αναζητηθεί στο Schaerf (1999) [1]. Ο όγκος των εργασιών που έχουν δημοσιευτεί έκτοτε σχετικά με το θέμα αυτό αυξάνεται σταθερά. Τα προβλήματα του εκπαιδευτικού χρονοπρογραμματισμού μπορούν να διακριθούν σε γενικές γραμμές ταξινομούνται σε τρεις κύριους τύπους:

- *Προβλήματα ωρολογίου προγράμματος Γυμνασίου - Λυκείου:*

Πρέπει να προγραμματιστούν σύνολα μαθητών που σχηματίζουν τάξεις σε ώρες διδασκαλίας που προϋποθέτουν συγκεκριμένη διαθεσιμότητα και εξειδίκευση των καθηγητών, ώστε να να δημιουργηθεί ένα εφικτό και ισορροπημένο ωρολόγιο πρόγραμμα. Το υποκείμενο μοντέλο έχει διατυπωθεί ως πρόβλημα χρωματισμού ακμών σε διμερές γράφημα. Στο πλαίσιο αυτής της διατύπωσης το αφηρημένο πρόβλημα μπορεί να είναι πολυωνυμικά επιλύσιμο. Παρ' όλα αυτά, η προσθήκη περιορισμών της πραγματικής ζωής το καθιστά NP-complete. Η διατύπωση του χρωματισμού ακμών του προβλήματος μπορεί να εντοπιστεί στις Csima (1971) [2] και Bondy & Murty (1976) [3] με την τελευταία να έχει αναφορές στις de Werra (1971) και Dempster (1971) [4].

- *Προβλήματα χρονοπρογραμματισμού πανεπιστημιακών μαθημάτων:*

[1] *NP (Nondeterministic Polynomial) problem: Μη-ντετερμινιστικό πολυώνυμο*

Ένα πρόβλημα ονομάζεται NP (μη ντετερμινιστικό πολυωνυμικό) εάν η λύση του μπορεί να μαντέψει και να επαληθευτεί σε πολυωνυμικό χρόνο (μη ντετερμινιστικό σημαίνει ότι δεν ακολουθείται κανένας συγκεκριμένος κανόνας για να γίνει η μαντεψιά). Εάν ένα πρόβλημα είναι NP και όλα τα άλλα NP προβλήματα μπορούν να αναχθούν σε πολυωνυμικό χρόνο σε αυτό, το πρόβλημα είναι NP-complete. Έτσι, η εύρεση ενός αποτελεσματικού αλγορίθμου για οποιοδήποτε NP-complete πρόβλημα συνεπάγεται ότι μπορεί να βρεθεί ένας αποτελεσματικός αλγόριθμος για όλα αυτά τα προβλήματα, δεδομένου ότι κάθε πρόβλημα που ανήκει σε αυτή την κατηγορία μπορεί να μετατραπεί σε οποιοδήποτε άλλο μέλος της κατηγορίας. Όταν πρέπει να επιλυθεί ένα NP-complete πρόβλημα, μια προσέγγιση είναι η χρήση ενός πολυωνυμικού αλγορίθμου για την προσέγγιση της λύσης- η απάντηση που θα προκύψει με αυτόν τον τρόπο δεν θα είναι απαραίτητα η βέλτιστη αλλά θα είναι αρκετά κοντά.

Το πρόβλημα αυτό μπορεί να θεωρηθεί ως εξειδίκευση του προηγούμενου τύπου με τη διαφορά ότι οι φοιτητές μπορούν να ανήκουν σε περισσότερες από μία τάξεις. Ο κύριος στόχος είναι η ελαχιστοποίηση των επικαλύψεων διαλέξεων που αφορούν τους ίδιους φοιτητές.

- *Προβλήματα χρονοδιαγράμματος εξετάσεων:*

Το πρόβλημα αυτό αφορά εξετάσεις που διεξάγονται από σύνολα μαθητών και ο στόχος είναι να μειωθούν οι περιπτώσεις μαθητών που πρέπει να λάβουν μέρος σε περισσότερες από μία εξετάσεις ταυτόχρονα και να κατανεμηθούν ομοιόμορφα οι εξετάσεις σε όλη την εξεταστική περίοδο.

Γενικά έχει δημοσιευθεί ένας σημαντικός αριθμός εργασιών σχετικά με το πρόβλημα του Γυμνασίου - Λυκείου και ειδικότερα όσον αφορά η ελληνική παραλλαγή. Ορισμένες αντιπροσωπευτικές εργασίες είναι Valouxis & Housos (2003) [5], Beligiannis, Moschopoulos, Kaperonis, & Likothanassis (2008) [6] και Birbas, Daskalaki, & Housos (2009) [7].

Σε πολλά σχολεία και πανεπιστήμια, ο προγραμματισμός του ωρολογίου προγράμματος των μαθημάτων και των εξετάσεων γίνεται χειροκίνητα, γεγονός που απαιτεί μεγάλη προσπάθεια και χρόνο. Επιπλέον, ο χειροκίνητος προγραμματισμός οδηγεί συχνά σε λάθη ή δεν είναι πάντα σε θέση να ικανοποιήσει όλους τους περιορισμούς του προβλήματος.

Ο προγραμματισμός γίνεται συνήθως επαναληπτικά, ξεκινώντας με ένα κενό ωρολόγιο πρόγραμμα στο οποίο σχεδιάζονται τα μαθήματα των διαφόρων ομάδων. Αυτό έχει ως αποτέλεσμα την ετερογενή ποιότητα των ωρολογίων για τους μαθητές των διαφόρων ομάδων (είναι ευκολότερο να παρακολουθούνται καλά τα πρώτα, αλλά γίνεται όλο και πιο περίπλοκο καθώς προστίθενται περισσότερες ομάδες). Το ίδιο ισχύει και για τους εκπαιδευτικούς.

Σε γενικές γραμμές, ένα χειροκίνητο σχολικό ωρολόγιο πρόγραμμα:

- Είναι μια χρονοβόρα διαδικασία και απαιτεί πολύ χρόνο και προσπάθεια από την ομάδα σχεδιασμού.
- Λόγω του μεγάλου χρόνου σχεδιασμού του ημερολογίου, δεν είναι δυνατή η δοκιμή διαφορετικών διαμορφώσεων ή βελτιώσεων, γεγονός που καθιστά τον προγραμματισμό ανελαστικό.
- Ως αποτέλεσμα, μπορεί να επηρεαστεί η ποιότητα των ωρολογίων των φοιτητών, με αποτέλεσμα ο φοιτητής να χάνει ακόμη και κάποια μαθήματα λόγω επικαλύψεων.
- Η ανελαστικότητα των ωρολογίων προγραμμάτων μπορεί να τα καταστήσει ασύμβατα με τη διαθεσιμότητα των εκπαιδευτικών, γεγονός που καθιστά τις διαδικασίες πρόσληψης και τις συνθήκες για τους εκπαιδευτικούς δύσκολες.

1.3 Ορολογίες

Για να γίνει πιο κατανοητό το πρόβλημα που έχουμε να αντιμετωπίσουμε πρέπει να δώσουμε ορισμούς στις διάφορες ορολογίες που θα χρησιμοποιηθούν.

Ως **τμήμα** ορίζεται ένα σύνολο μαθητών στο οποίο διδάσκεται ένα συγκεκριμένο μάθημα.

Ως **τάξη** ορίζεται το επίπεδο ενός τμήματος, δηλαδή για παράδειγμα Ε' Δημοτικού, Β΄ Γυμνασίου κλπ.

Ως **αντικείμενο** ορίζεται το περιεχόμενο του μαθήματος που διδάσκεται.

Ως **κύκλος σπουδών** ορίζεται το σύνολο των αντικείμενων που πρέπει να διδαχθεί σε κάθε τάξη.

Ως **περίοδος** ορίζεται μια θυρίδα στο ωρολόγιο πρόγραμμα κατά την οποία μπορεί να διδαχθεί ένα αντικείμενο. Η διάρκεια μιας περιόδου μπορεί να αλλάζει από τάξη σε τάξη, ακόμα και ανά περίοδο. Μια περίοδος ονομάζεται κενή όταν δεν γίνεται κάποια διδασκαλία. Για να είναι συμπαγές ένα ωρολόγιο πρόγραμμα δεν πρέπει να έχει κενές περιόδους.

1.4 Περιορισμοί

Για να υλοποιηθεί το βέλτιστο σχολικό πρόγραμμα πρέπει να ληφθεί υπόψιν μια πληθώρα παραμέτρων και πρέπει να ικανοποιεί ένα μεγάλο πλήθος περιορισμών. Σαν βέλτιστη λύση του προβλήματος ορίζουμε την δημιουργία ενός ωρολογίου προγράμματος στο οποίο οι συνδυασμοί καθηγητή – τμήματος τοποθετούνται στις περιόδους με τέτοιο τρόπο, ώστε να ικανοποιούνται οι ανελαστικοί και οι ελαστικοί περιορισμοί [6].

Προφανώς η δημιουργία ενός σχολικού προγράμματος διαφέρει από χώρα σε χώρα αφού αλλάζει το εκπαιδευτικό σύστημα. Στην παρούσα εργασία ασχολούμαστε με το εκπαιδευτικό σύστημα τη Ελλάδας και συγκεκριμένα με την Πρωτοβάθμια και τη Δευτεροβάθμια εκπαίδευση. Οι ανελαστικοί περιορισμοί που θα τεθούν θα είναι απαραίτητο να ικανοποιούνται όλοι ανεξαιρέτως για να μπορούμε να ισχυριστούμε ότι το πρόγραμμα είναι ρεαλιστικό και μπορεί να εφαρμοστεί. Οι ελαστικοί περιορισμοί από την άλλη είναι καλό α ικανοποιούνται γιατί θα μας προσφέρουν ένα βελτιωμένο πρόγραμμα που θα ικανοποιεί περισσότερους περιορισμούς, άρα θα είναι και πιο ποιοτικό, όμως σπάνια μπορούν να ικανοποιηθούν όλοι οι περιορισμοί καθώς έρχονται συχνά σε συγκρούσεις.

Στην παρούσα εργασία θα έχουμε 4 περιορισμούς οι οποίοι θεωρούνται όλοι ανελαστικοί, οπότε θα πρέπει να ικανοποιούνται όλοι για να δημιουργηθεί ένα ωρολόγιο πρόγραμμα.

Περιορισμός 1: Κάθε Τμήμα πρέπει να έχει τον απαραίτητο αριθμό μαθημάτων που προσδιορίζεται από τον Κύκλο Σπουδών.

Περιορισμός 2: Κάθε Τμήμα μπορεί να κάνει μόνο ένα μάθημα σε κάθε Περίοδο.

Περιορισμός 3: Ο Καθηγητής μπορεί να διδάσκει το πολύ σε ένα Τμήμα τη φορά.

Περιορισμός 4: Μέγιστες ώρες εργασίας κάθε Καθηγητή.

Περιορισμός 5: Ο ίδιος Καθηγητής κάνει σε μια Τάξη όλες τις ώρες ενός συγκεκριμένου μαθήματος.

1.5 Παρόμοιες εργασίες

Σε αυτή την ενότητα θα παρουσιάσουμε ορισμένες εισηγήσεις που έχουν επιχειρήσει να επιλύσουν το πρόβλημα της εύρεσης βέλτιστου ωρολογίου σχολικού προγράμματος και τους αλγορίθμους που χρησιμοποίησαν.

Στην εισήγηση των Tassopoulos και Beligiannis (2012) [8], εφαρμόζεται ένας υβριδικός αλγόριθμος βελτιστοποίησης σμήνους σωματιδίων (PSO)[2] σε προβλήματα ωρολογίου προγράμματος γυμνασίου. Ο προτεινόμενος αλγόριθμος PSO χρησιμοποιείται για τη δημιουργία εφικτών και αποτελεσματικών ωρολογίων προγραμμάτων γυμνασίων. Προκειμένου να αποδειχθεί η αποτελεσματικότητα του προτεινόμενου αλγορίθμου που βασίζεται στο PSO, πραγματοποιήθηκαν πειράματα με δεδομένα εισόδου από τον πραγματικό κόσμο που προέρχονται από πολλά διαφορετικά ελληνικά λύκεια. Τα υπολογιστικά αποτελέσματα δείχνουν ότι ο προτεινόμενος υβριδικός αλγόριθμος βασισμένος στο PSO αποδίδει καλύτερα από τις υπάρχουσες προσεγγίσεις που εφαρμόζονται στις ίδιες περιπτώσεις εισόδου ωρολογίων προγραμμάτων σχολείων με τα ίδια κριτήρια αξιολόγησης.

Οι Skoullis, Tassopoulos, & Beligiannis (2017) [9] στην εισήγηση τους παρουσιάζουν την εφαρμογή ενός υβριδικού αλγορίθμου βελτιστοποίησης σμήνους αιλουροειδών (CSO)[3] για την επίλυση του προβλήματος του σχολικού ωρολογίου προγράμματος. Αυτός ο εύχρηστος, αποδοτικός και γρήγορος αλγόριθμος είναι μια υβριδική παραλλαγή του κλασικού αλγορίθμου CSO. Η αποδοτικότητα και η απόδοσή του αποδεικνύεται με τη διεξαγωγή πειραμάτων με δεδομένα εισόδου από τον πραγματικό κόσμο. Τα δεδομένα αυτά, τα οποία συλλέχθηκαν από διάφορα λύκεια στην Ελλάδα, έχουν επίσης χρησιμοποιηθεί ως δοκιμαστικά παραδείγματα από πολλούς άλλους ερευνητές στις δημοσιεύσεις τους. Τα αποτελέσματα αποκαλύπτουν ότι αυτός ο υβριδικός αλγόριθμος που βασίζεται στην CSO, εφαρμοζόμενος στις ίδιες δοκιμαστικές περιπτώσεις ωρολογίου προγράμματος σχολείων με τα ίδια κριτήρια αξιολόγησης,

[2] Particle Swarm Optimization (PSO) είναι μια υπολογιστική μέθοδος που βελτιστοποιεί ένα πρόβλημα προσπαθώντας επαναληπτικά να βελτιώσει μια υποψήφια λύση σε σχέση με ένα δεδομένο μέτρο ποιότητας. Επιλύει ένα πρόβλημα διαθέτοντας έναν πληθυσμό υποψήφιων λύσεων, που εδώ ονομάζονται σωματίδια, και μετακινώντας αυτά τα σωματίδια στο χώρο αναζήτησης σύμφωνα με έναν απλό μαθηματικό τύπο για τη θέση και την ταχύτητα του σωματιδίου. Η μετακίνηση κάθε σωματιδίου επηρεάζεται από την καλύτερη γνωστή τοπική θέση του, αλλά καθοδηγείται επίσης προς τις καλύτερες γνωστές θέσεις στο χώρο αναζήτησης, οι οποίες ενημερώνονται καθώς άλλα σωματίδια βρίσκουν καλύτερες θέσεις. Αυτό αναμένεται να οδηγήσει το σμήνος προς τις καλύτερες λύσεις.

[3] Cat Swarm Optimization (CSO). Το CSO δημιουργήθηκε παρατηρώντας τις συμπεριφορές των γατών και αποτελείται από δύο υπομοντέλα, δηλαδή τη λειτουργία εντοπισμού και τη λειτουργία αναζήτησης, τα οποία διαμορφώνουν τις συμπεριφορές των γατών. Τα πειραματικά αποτελέσματα με τη χρήση έξι δοκιμαστικών συναρτήσεων καταδεικνύουν ότι η CSO έχει πολύ καλύτερες επιδόσεις από τη βελτιστοποίηση σμήνους σωματιδίων (PSO)

παρουσιάζει καλύτερη απόδοση σε μικρότερο υπολογιστικό χρόνο σε σύγκριση με την πλειονότητα των άλλων υφιστάμενων προσεγγίσεων. Η κύρια διαδικασία του αλγορίθμου αποτελεί μια παραλλαγή του κλασικού αλγορίθμου CSO, κατάλληλα τροποποιημένου ώστε να εφαρμοστεί για την επίλυση του προβλήματος του σχολικού ωρολογίου προγράμματος. Η διαδικασία αυτή περιέχει τις κύριες αλγοριθμικές διαφορές της προτεινόμενης προσέγγισης σε σχέση με άλλους αλγορίθμους που παρουσιάζονται στην αντίστοιχη βιβλιογραφία.

Οι Demirovic & Musliu (2017) [10] στην εισήγηση τους προτείνουν μια νέα προσέγγιση μοντελοποίησης για την εύρεση ωρολογίου σχολικού προγράμματος με τη χρήση διανυσμάτων bit, στην οποία το κόστος περιορισμών μπορεί να υπολογιστεί με τη χρήση πράξεων bit. Αυτό το μοντέλο επιτρέπει τον αποδοτικό υπολογισμό του κόστους περιορισμών καθιστώντας το χρήσιμο κατά την υλοποίηση αλγορίθμων εύρεσης ωρολογίου σχολικού προγράμματος. Επιπλέον, μπορεί να χρησιμοποιηθεί για την επίλυση της εύρεσης βέλτιστου ωρολογίου σχολικού προγράμματος με λύτες ικανοποιησιμότητας modulo theory (SMT) που υποστηρίζουν διανύσματα bit. Αξιολογώντας τις επιδόσεις για την προσέγγιση μοντελοποίησης διανυσμάτων bit και συγκρίνοντάς τες με την κορυφαία μηχανή KHE [11] κατά την ανάπτυξη αλγορίθμων τοπικής αναζήτησης, όπως η αναρρίχηση λόφου και η προσομοιωμένη ανόπτηση. Από τα πειράματα αναδείχτηκε ότι η προσέγγισή τους είναι χρήσιμη για αυτό το πρόβλημα και ξεπέρασε σε απόδοση πολλές σύγχρονες μεθόδους.

Οι Saviniec, Santos, & Costa (2017) [12] ισχυρίστηκαν ότι οι περιπτώσεις μεσαίου και μεγάλου μεγέθους εξακολουθούν να μην επιλύονται αποτελεσματικά από τα προγράμματα με τη χρήση σύγχρονων επιλυτών και η επιστημονική κοινότητα έχει δώσει ιδιαίτερη προσοχή στην επινόηση εναλλακτικών αλγορίθμων μαλακού υπολογισμού. Στην εργασία τους προτείνουν μια προσέγγιση ήπιας υπολογιστικής που βασίζεται σε μεταευρετικά πλαίσια επαναληπτικής τοπικής αναζήτησης και μεταβλητής αναζήτησης γειτονιάς. Οι αλγόριθμοί τους ενσωματώνουν νέες δομές γειτονιάς και ρουτίνες τοπικής αναζήτησης για την εκτέλεση μιας αποτελεσματικής αναζήτησης. Επικύρωσαν τους προτεινόμενους αλγορίθμους σε παραλλαγές του προβλήματος χρησιμοποιώντας επτά δημόσιες περιπτώσεις και ένα νέο σύνολο δεδομένων με 34 πραγματικές περιπτώσεις, συμπεριλαμβανομένων μεγάλων περιπτώσεων. Τα αποτελέσματα καταδεικνύουν ότι οι προτεινόμενοι αλγόριθμοι υπερτερούν των σύγχρονων προσεγγίσεων και στις δύο περιπτώσεις, βρίσκοντας τις καλύτερες λύσεις σε 38 από τις 41 δοκιμασμένες περιπτώσεις.

Οι Fonseca, Santos & Carrano (2016) [13] εφάρμοσαν την μέθοδο αναρρίχησης λόφου καθυστερημένης αποδοχής (LAHC)[4] για την επίλυση του προβλήματος του σχολικού ωρολογίου προγράμματος. Ο αρχικός αλγόριθμος και οι δύο παραλλαγές που προτείνονται δοκιμάστηκαν από κοινού με άλλες σύγχρονες μεθόδους για την επίλυση των περιπτώσεων που προτάθηκαν στον Τρίτο Διεθνή Διαγωνισμό Ωρολογίου Προγράμματος. Ακολουθώντας τους ίδιους κανόνες του διαγωνισμού, οι αλγόριθμοι που βασίζονται στον LAHC υπερέβησαν αισθητά τις νικήτριες μεθόδους. Αυτά τα αποτελέσματα, καθώς και αναφορές από τη

[4] Late acceptance hill climbing (LAHC) δημιουργήθηκε από τον Yuri Bykov το 2008 και είναι μια μεταευρετική μέθοδος αναζήτησης που χρησιμοποιεί μεθόδους τοπικής αναζήτησης που χρησιμοποιούνται για μαθηματική βελτιστοποίηση.

βιβλιογραφία, υποδηλώνουν ότι η LAHC είναι μια αξιόπιστη μέθοδος που μπορεί να ανταγωνιστεί τους πιο συχνά χρησιμοποιούμενους αλγορίθμους τοπικής αναζήτησης.

Κεφάλαιο 2ο: Προβλήματα Ικανοποίησης Περιορισμών (CSP)

2.1 Τι είναι τα Προβλήματα Ικανοποίησης Περιορισμών;

Τα προβλήματα ικανοποίησης περιορισμών (CSPs) είναι μαθηματικά ερωτήματα που ορίζονται ως ένα σύνολο αντικειμένων των οποίων η κατάσταση πρέπει να ικανοποιεί έναν αριθμό περιορισμών ή κανόνων. Τα CSPs αναπαριστούν τις μονάδες ενός προβλήματος ως μια ομοιογενή συλλογή πεπερασμένων περιορισμών επί μεταβλητών, η οποία επιλύεται με μεθόδους ικανοποίησης περιορισμών. Τα CSPs αποτελούν αντικείμενο έρευνας τόσο στην τεχνητή νοημοσύνη όσο και στην επιχειρησιακή έρευνα, δεδομένου ότι η κανονικότητα στη διατύπωσή τους παρέχει μια κοινή βάση για την ανάλυση και επίλυση προβλημάτων πολλών φαινομενικά άσχετων ομάδων. Τα CSPs συχνά παρουσιάζουν υψηλή πολυπλοκότητα, απαιτώντας ένα συνδυασμό ευρετικών και συνδυαστικών μεθόδων αναζήτησης για να επιλυθούν σε εύλογο χρόνο. Ο προγραμματισμός περιορισμών (CP) είναι το πεδίο της έρευνας που επικεντρώνεται ειδικά στην αντιμετώπιση τέτοιου είδους προβλημάτων. [14]

Επιπλέον, τα πρόβλημα ικανοποιησιμότητας Boolean (SAT), η ικανοποιησιμότητα modulo θεωριών (SMT), ο μικτός ακέραιος προγραμματισμός (MIP) και ο προγραμματισμός απαντητικών συνόλων (ASP) είναι όλα πεδία έρευνας που εστιάζουν στην επίλυση συγκεκριμένων μορφών του προβλήματος ικανοποίησης περιορισμών .

Παραδείγματα προβλημάτων που μπορούν να μοντελοποιηθούν ως πρόβλημα ικανοποίησης περιορισμών περιλαμβάνουν:

- Συμπερασματολογία τύπου [15]
- Γρίφος των οκτώ βασιλισσών
- Πρόβλημα χρωματισμού χάρτη
- Πρόβλημα μέγιστης αποκοπής [16]
- Sudoku, Σταυρόλεξα, Futoshiki, Kakuro (Cross Sums), Numbrix, Hidato και πολλοί άλλοι λογικοί γρίφοι

Οι μοντελοποιήσεις συχνά παρέχονται με οδηγούς των λυτών CP, ASP, Boolean SAT και SMT. Στη γενική περίπτωση, τα προβλήματα περιορισμών μπορεί να είναι πολύ πιο δύσκολα, και μπορεί να μην είναι εκφραζόμενα σε ορισμένα από αυτά τα απλούστερα συστήματα.

Παραδείγματα της "πραγματικής ζωής" περιλαμβάνουν τον αυτοματοποιημένο προγραμματισμό [17], τη λεξιλογική αποσαφήνιση [18], τη μουσικολογία [19], τη διαμόρφωση προϊόντων [20] και την κατανομή πόρων [21].

Η ύπαρξη μιας λύσης σε ένα CSP μπορεί να θεωρηθεί ως πρόβλημα απόφασης. Αυτό μπορεί να αποφασιστεί με την εύρεση μιας λύση, ή την αποτυχία εύρεσης λύσης μετά από

ολοκληρωτική αναζήτηση (οι στοχαστικοί αλγόριθμοι τυπικά δεν φτάνουν ποτέ σε ένα ολοκληρωτικό συμπέρασμα, ενώ οι κατευθυνόμενες αναζητήσεις συχνά καταλήγουν, σε επαρκώς μικρά προβλήματα). Σε ορισμένες περιπτώσεις το CSP μπορεί να είναι γνωστό ότι έχει λύσεις εκ των προτέρων, μέσω κάποιας άλλης μαθηματικής διαδικασίας εξαγωγής συμπερασμάτων.

2.2 Ορισμός Προβλημάτων Ικανοποίησης Περιορισμών

Τυπικά, ένα πρόβλημα ικανοποίησης περιορισμών ορίζεται ως μια τριάδα $\langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{C} \rangle$, όπου:

- $\boldsymbol{X} = \{\boldsymbol{X_1}, \dots, \boldsymbol{X_v}\}$ είναι ένα σύνολο μεταβλητών,
- $\boldsymbol{D} = \{\boldsymbol{D_1}, \dots, \boldsymbol{D_v}\}$ είναι ένα σύνολο από τα αντίστοιχα πεδία τιμών τους,
- $\boldsymbol{C} = \{\boldsymbol{C_1}, \dots, \boldsymbol{C_v}\}$ είναι ένα σύνολο περιορισμών.

Κάθε μεταβλητή $\boldsymbol{X_i}$ μπορεί να πάρει τιμές στο μη-κενό σύνολο $\boldsymbol{D_i}$. Κάθε περιορισμός $\boldsymbol{C_j} \in \boldsymbol{C}$ είναι με τη σειρά του ένα ζεύγος $\langle \boldsymbol{t_j}, \boldsymbol{R_j} \rangle$, όπου $\boldsymbol{t_j} \subset \boldsymbol{X}$ είναι ένα υποσύνολο των $\boldsymbol{k}$ μεταβλητών και $\boldsymbol{R_j}$ είναι $\boldsymbol{k}$-ιοστή σχέση στο αντίστοιχο υποσύνολο των συνόλων $\boldsymbol{D_j}$. Μια αξιολόγηση των μεταβλητών είναι μια συνάρτηση από ένα υποσύνολο μεταβλητών σε ένα συγκεκριμένο σύνολο τιμών στο αντίστοιχο υποσύνολο των πεδίων τιμών τους. Μια εκτίμηση $\boldsymbol{v}$ ικανοποιεί έναν περιορισμό $\langle \boldsymbol{t_j}, \boldsymbol{R_j} \rangle$ εάν οι τιμές που αποδίδονται στις μεταβλητές $\boldsymbol{t_j}$ ικανοποιούν τη σχέση $\boldsymbol{R_j}$. [22]

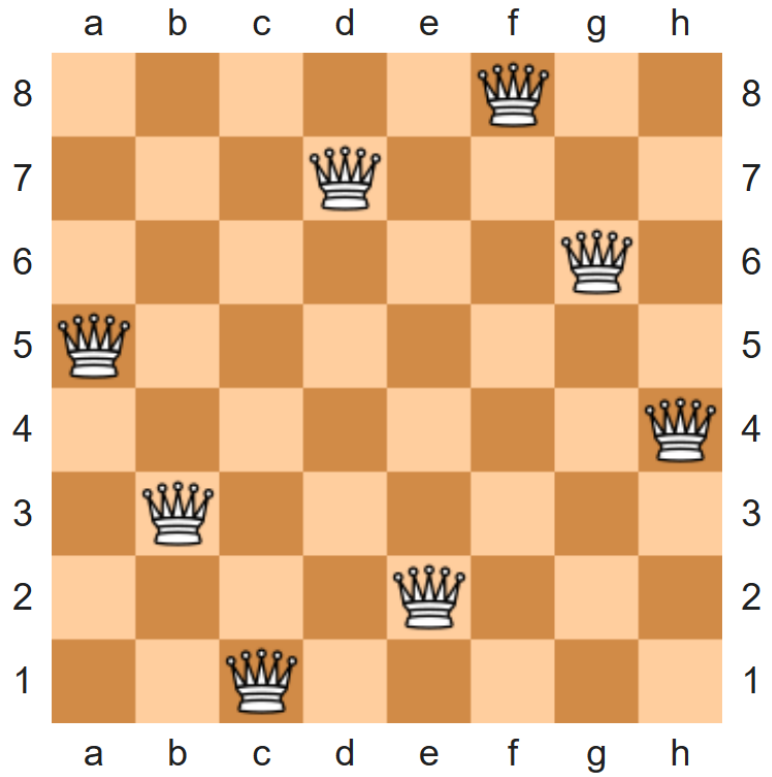
Μια εκτίμηση είναι συνεπής εάν δεν παραβιάζει κανέναν από τους περιορισμούς. Μια εκτίμηση είναι πλήρης εάν περιλαμβάνει όλες τις μεταβλητές. Μια εκτίμηση είναι λύση αν είναι ταυτόχρονα συνεπής και πλήρης, πρακτικά μια τέτοια εκτίμηση λέγεται ότι «*λύνει*» το πρόβλημα ικανοποίησης περιορισμών.

2.3 Παραδείγματα Προβλημάτων Ικανοποίησης Περιορισμών

2.3.1 Γρίφος των οκτώ βασιλισσών

Ο γρίφος των οκτώ βασιλισσών είναι το πρόβλημα της τοποθέτησης οκτώ σκακιστικών βασιλισσών σε μια σκακιέρα 8×8 έτσι ώστε καμία από τις δύο βασίλισσες να μην απειλεί η μία την άλλη. Συνεπώς, η λύση απαιτεί να μην υπάρχουν δύο βασίλισσες που να μοιράζονται την ίδια γραμμή, στήλη ή διαγώνιο. Υπάρχουν 92 λύσεις. Το πρόβλημα τέθηκε για πρώτη φορά στα μέσα του 19ου αιώνα. Στη σύγχρονη εποχή, χρησιμοποιείται συχνά ως παράδειγμα προβλήματος για διάφορες τεχνικές προγραμματισμού υπολογιστών.

Ο γρίφος των οκτώ βασιλισσών είναι μια ειδική περίπτωση του γενικότερου προβλήματος των n βασιλισσών για την τοποθέτηση n μη επιτιθέμενων βασιλισσών σε μια σκακιέρα $n \times n$. Λύσεις υπάρχουν για όλους τους φυσικούς αριθμούς n με εξαίρεση τους $n = 2$ και $n = 3$. Αν και ο ακριβής αριθμός των λύσεων είναι γνωστός μόνο για $n \leq 27$, ο ασυμπτωτικός ρυθμός αύξησης του αριθμού των λύσεων είναι περίπου $(0{,}143 \cdot n)^n$.



Εικόνα 1: Η μόνη συμμετρική λύση στο παζλ των οκτώ βασιλισσών

Ιστορία του γρίφου

Ο σκακιστής Max Bezzel δημοσίευσε το γρίφο των οκτώ βασιλισσών το 1848. Ο Franz Nauck δημοσίευσε τις πρώτες λύσεις το 1850. Ο Nauck επέκτεινε επίσης το γρίφο στο πρόβλημα των n βασιλισσών, με n βασίλισσες σε μια σκακιέρα $n \times n$ τετραγώνων.

Έκτοτε, πολλοί μαθηματικοί, συμπεριλαμβανομένου του Carl Friedrich Gauss (ένας εκ των σπουδαιότερων μαθηματικών), έχουν εργαστεί τόσο πάνω στον γρίφο των οκτώ βασιλισσών όσο και στη γενικευμένη εκδοχή του με n βασίλισσες. Το 1874, ο S. Gunther πρότεινε μια μέθοδο που χρησιμοποιούσε προσδιοριστές για την εύρεση λύσεων και ο J.W.L. Glaisher βελτίωσε την προσέγγιση του Gunther.

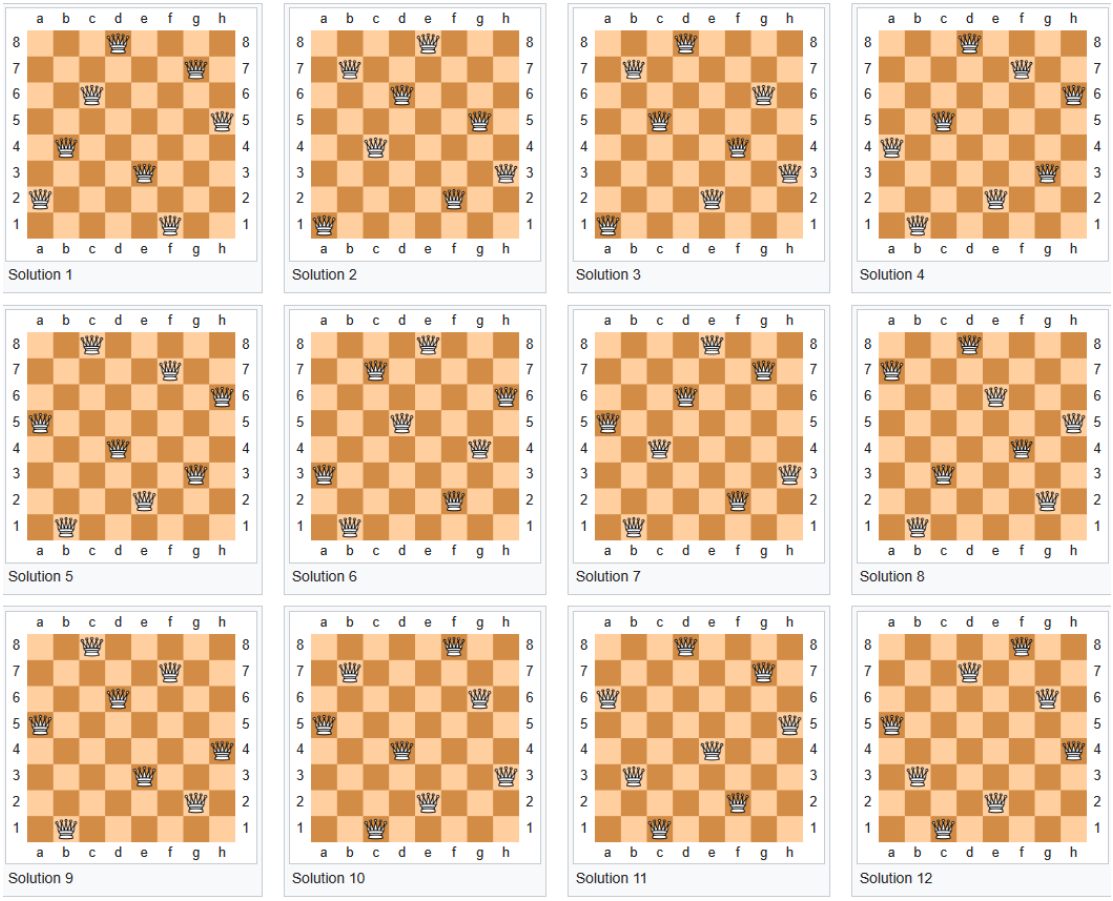
Το 1972, ο Edsger Dijkstra χρησιμοποίησε αυτό το πρόβλημα για να δείξει τη δύναμη αυτού που ονόμασε δομημένο προγραμματισμό. Δημοσίευσε μια πολύ λεπτομερή περιγραφή ενός αλγορίθμου οπισθοδρόμησης με βάση το βάθος.

Κατασκευή και καταμέτρηση λύσεων όταν $n = 8$

Το πρόβλημα της εύρεσης όλων των λύσεων στο πρόβλημα των 8 βασιλισσών μπορεί να είναι αρκετά δαπανηρό από υπολογιστική άποψη, καθώς υπάρχουν 4.426.165.368 πιθανές

διατάξεις οκτώ βασιλισσών σε ένα ταμπλό 8×8, αλλά μόνο 92 λύσεις. Είναι δυνατόν να χρησιμοποιηθούν συντομεύσεις που μειώνουν τις υπολογιστικές απαιτήσεις ή κανόνες που αποφεύγουν τις υπολογιστικές τεχνικές «ωμής βίας». Για παράδειγμα, εφαρμόζοντας έναν απλό κανόνα που επιλέγει μία βασίλισσα από κάθε στήλη, είναι δυνατόν να μειωθεί ο αριθμός των δυνατοτήτων σε 16.777.216 (δηλαδή 88) δυνατούς συνδυασμούς. Η δημιουργία μεταθέσεων μειώνει περαιτέρω τις δυνατότητες σε μόλις 40.320 (δηλαδή 8!), οι οποίες μπορούν στη συνέχεια να ελεγχθούν για διαγώνιες επιθέσεις.

Ο γρίφος των οκτώ βασιλισσών έχει 92 διαφορετικές λύσεις. Αν οι λύσεις που διαφέρουν μόνο από τις πράξεις συμμετρίας της περιστροφής και της αντανάκλασης του πίνακα υπολογίζονται ως μία, το παζλ έχει 12 λύσεις. Αυτές ονομάζονται θεμελιώδεις λύσεις, εκπρόσωποι της καθεμιάς παρουσιάζονται παρακάτω στην Εικ. 2.



Εικόνα 2: Όλες οι πιθανές λύσεις του γρίφου των 8 βασιλισσών

Μια θεμελιώδης λύση έχει συνήθως οκτώ παραλλαγές (συμπεριλαμβανομένης της αρχικής της μορφής) που λαμβάνονται με περιστροφή 90°, 180° ή 270° και στη συνέχεια με αντανάκλαση καθεμιάς από τις τέσσερις περιστροφικές παραλλαγές σε καθρέφτη σε σταθερή θέση. Ωστόσο, μία από τις 12 θεμελιώδεις λύσεις (λύση 12 παρακάτω) είναι πανομοιότυπη με τη δική της περιστροφή κατά 180°, οπότε έχει μόνο τέσσερις παραλλαγές (η ίδια και η

αντανάκλασή της, η περιστροφή της κατά 90° και η αντανάκλασή της). Τέτοιες λύσεις έχουν μόνο δύο παραλλαγές (η ίδια και η αντανάκλασή της). Έτσι, ο συνολικός αριθμός των διαφορετικών λύσεων είναι $11 \times 8 + 1 \times 4 = 92$.

Εύρεση λύσεων μέσω αλγορίθμων

Η εύρεση όλων των λύσεων στο γρίφο των οκτώ βασιλισσών είναι ένα καλό παράδειγμα ενός απλού αλλά μη τετριμμένου προβλήματος. Για το λόγο αυτό, χρησιμοποιείται συχνά ως παράδειγμα προβλήματος για διάφορες τεχνικές προγραμματισμού, συμπεριλαμβανομένων μη παραδοσιακών προσεγγίσεων όπως ο προγραμματισμός με περιορισμούς, ο λογικός προγραμματισμός ή οι γενετικοί αλγόριθμοι. Τις περισσότερες φορές, χρησιμοποιείται ως παράδειγμα προβλήματος που μπορεί να επιλυθεί με αναδρομικό αλγόριθμο, διατυπώνοντας το πρόβλημα των n βασιλισσών επαγωγικά ως προς την προσθήκη μιας βασίλισσας σε οποιαδήποτε λύση του προβλήματος της τοποθέτησης $n - 1$ βασιλισσών σε μια σκακιέρα $n \times n$. Η επαγωγή καταλήγει στη λύση του "προβλήματος" της τοποθέτησης 0 βασιλισσών στη σκακιέρα, η οποία είναι η άδεια σκακιέρα [23].

Αυτή η τεχνική μπορεί να χρησιμοποιηθεί με τρόπο που είναι πολύ πιο αποδοτικός από τον αφελή αλγόριθμο «ωμής» αναζήτησης, ο οποίος εξετάζει όλες τις $64^8 = 2\text{^}48 = 281.474.976.710.656$ πιθανές τυφλές τοποθετήσεις οκτώ βασιλισσών και στη συνέχεια τις φιλτράρει για να αφαιρέσει όλες τις τοποθετήσεις που τοποθετούν δύο βασίλισσες είτε στο ίδιο τετράγωνο (αφήνοντας μόνο $64!/56! = 178.462.987.637.760$ πιθανές τοποθετήσεις) είτε σε θέσεις που αλληλοεπιτίθενται. Αυτός ο πολύ φτωχός αλγόριθμος θα παράγει, μεταξύ άλλων, τα ίδια αποτελέσματα ξανά και ξανά σε όλες τις διαφορετικές μεταθέσεις των τοποθετήσεων των οκτώ βασιλισσών, καθώς και επαναλαμβάνοντας τους ίδιους υπολογισμούς ξανά και ξανά για τα διαφορετικά υποσύνολα κάθε λύσης. Ένας καλύτερος αλγόριθμος «ωμής βίας» τοποθετεί μία μόνο βασίλισσα σε κάθε σειρά, οδηγώντας σε μόνο $8^8 = 2^{24} = 16.777.216$ τυφλές τοποθετήσεις [24].

Είναι δυνατόν να υπάρξουν πολύ καλύτεροι τρόποι από αυτόν. Ένας αλγόριθμος λύνει το γρίφο των οκτώ βασιλισσών δημιουργώντας τις μεταθέσεις των αριθμών 1 έως 8 (από τις οποίες υπάρχουν $8! = 40.320$) και χρησιμοποιεί τα στοιχεία κάθε μεταθέσεως ως δείκτες για να τοποθετήσει μια βασίλισσα σε κάθε σειρά. Στη συνέχεια, απορρίπτει τις σκακιέρες με διαγώνιες θέσεις επίθεσης [25].

Το πρόγραμμα αναδρομικής αναζήτησης βάθους, μια μικρή βελτίωση της μεθόδου αντιμετάθεσης, κατασκευάζει το δέντρο αναζήτησης εξετάζοντας μία γραμμή του πίνακα κάθε φορά, εξαλείφοντας τις περισσότερες θέσεις του πίνακα που δεν επιλύονται σε πολύ πρώιμο στάδιο της κατασκευής τους. Επειδή απορρίπτει τις επιθέσεις πύργου και διαγωνίου ακόμη και σε ελλιπείς σκακιέρες, εξετάζει μόνο 15.720 πιθανές τοποθετήσεις βασίλισσας. Μια περαιτέρω βελτίωση, η οποία εξετάζει μόνο 5.508 πιθανές τοποθετήσεις βασιλισσών, είναι ο συνδυασμός της μεθόδου που βασίζεται στη μεταστοιχείωση με τη μέθοδο πρώιμης περικοπής: οι μεταστοιχειώσεις παράγονται σε βάθος και ο χώρος αναζήτησης περικόπτεται εάν η μερική μεταστοιχείωση παράγει μια διαγώνια επίθεση. Ο προγραμματισμός περιορισμών μπορεί επίσης να είναι πολύ αποτελεσματικός σε αυτό το πρόβλημα [25].

Μια εναλλακτική λύση στην εξαντλητική αναζήτηση είναι ένας αλγόριθμος "επαναληπτικής επιδιόρθωσης", ο οποίος συνήθως ξεκινάει με όλες τις βασίλισσες στο ταμπλό, για παράδειγμα με μία βασίλισσα ανά στήλη. Στη συνέχεια μετράει τον αριθμό των συγκρούσεων (επιθέσεων) και χρησιμοποιεί μια ευρετική μέθοδο για να καθορίσει πώς να βελτιώσει την τοποθέτηση των βασιλισσών. Η ευρετική μέθοδος "ελάχιστων συγκρούσεων", μετακινώντας το κομμάτι με τον μεγαλύτερο αριθμό συγκρούσεων στο τετράγωνο της ίδιας στήλης όπου ο αριθμός των συγκρούσεων είναι ο μικρότερος, είναι ιδιαίτερα αποτελεσματική: βρίσκει εύκολα λύση ακόμη και στο πρόβλημα των 1.000.000 βασιλισσών [23].

Σε αντίθεση με την αναδρομική αναζήτηση που περιγράφηκε παραπάνω, η επαναληπτική επιδιόρθωση δεν εγγυάται λύση: όπως όλες οι άπληστες διαδικασίες, μπορεί να κολλήσει σε ένα τοπικό βέλτιστο. (Σε μια τέτοια περίπτωση, ο αλγόριθμος μπορεί να επανεκκινήσει με διαφορετική αρχική παραμετροποίηση). Από την άλλη πλευρά, μπορεί να επιλύσει προβλήματα μεγέθους που είναι αρκετές τάξεις μεγέθους πέραν του πεδίου εφαρμογής μιας αναζήτησης σε βάθος [25].

Ως εναλλακτική λύση στην αναδρομή, οι λύσεις μπορούν να καταμετρηθούν με αναδρομική απαρίθμηση έγκυρων μερικών λύσεων, μία γραμμή κάθε φορά. Αντί να κατασκευάζονται ολόκληρες θέσεις του πίνακα, οι μπλοκαρισμένες διαγώνιες και στήλες παρακολουθούνται με πράξεις κατά δυφία. Αυτό δεν επιτρέπει την ανάκτηση μεμονωμένων λύσεων [26].

Το παρακάτω πρόγραμμα, είναι μια μετάφραση της λύσης του Niklaus Wirth (2004) [27] στη γλώσσα προγραμματισμού Python, αλλά δεν χρησιμοποιεί την αριθμητική των δεικτών που υπάρχει στο πρωτότυπο και χρησιμοποιεί λίστες για να κρατήσει τον κώδικα του προγράμματος όσο το δυνατόν πιο απλό. Με τη χρήση μιας coroutine με τη μορφή μιας γεννήτριας συνάρτησης, και οι δύο εκδοχές του πρωτοτύπου μπορούν να ενοποιηθούν για τον

```
"""OR-Tools solution to the N-queens problem."""
import sys
import time
from ortools.sat.python import cp_model


class NQueenSolutionPrinter(cp_model.CpSolverSolutionCallback):
    """Print intermediate solutions."""

    def __init__(self, queens):
        cp_model.CpSolverSolutionCallback.__init__(self)
        self.__queens = queens
        self.__solution_count = 0
        self.__start_time = time.time()

    def solution_count(self):
        return self.__solution_count

    def on_solution_callback(self):
        current_time = time.time()
        print('Solution %i, time = %f s' %
              (self.__solution_count, current_time - self.__start_time))
        self.__solution_count += 1

        all_queens = range(len(self.__queens))
        for i in all_queens:
            for j in all_queens:
                if self.Value(self.__queens[j]) == i:
                    # There is a queen in column j, row i.
                    print('Q', end=' ')
```

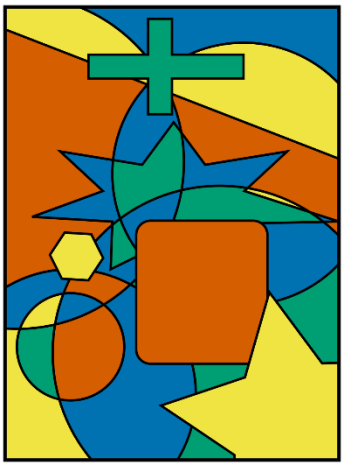
υπολογισμό είτε μιας είτε όλων των λύσεων. Εξετάζονται μόνο 15.720 πιθανές τοποθετήσεις βασιλισσών.

```
                else:
                    print('_', end=' ')
            print()
        print()


def main(board_size):
    # Creates the solver.
    model = cp_model.CpModel()

    # Creates the variables.
    # There are `board_size` number of variables, one for a queen in each column
    # of the board. The value of each variable is the row that the queen is in.
    queens = [
        model.NewIntVar(0, board_size - 1, 'x%i' % i) for i in range(board_size)
    ]

    # Creates the constraints.
    # All rows must be different.
    model.AddAllDifferent(queens)

    # No two queens can be on the same diagonal.
    model.AddAllDifferent(queens[i] + i for i in range(board_size))
    model.AddAllDifferent(queens[i] - i for i in range(board_size))

    # Solve the model.
    solver = cp_model.CpSolver()
    solution_printer = NQueenSolutionPrinter(queens)
    solver.parameters.enumerate_all_solutions = True
    solver.Solve(model, solution_printer)

    # Statistics.
    print('\nStatistics')
    print(f'  conflicts      : {solver.NumConflicts()}')
    print(f'  branches       : {solver.NumBranches()}')
    print(f'  wall time      : {solver.WallTime()} s')
    print(f'  solutions found: {solution_printer.solution_count()}')


if __name__ == '__main__':
    # By default, solve the 8x8 problem.
    size = 8
    if len(sys.argv) > 1:
        size = int(sys.argv[1])
    main(size)
```

2.3.2 Θεώρημα των τεσσάρων χρωμάτων (Four color theorem)

Στα μαθηματικά, το θεώρημα των τεσσάρων χρωμάτων, ή το θεώρημα των τεσσάρων χρωμάτων του χάρτη, ορίζει ότι δεν απαιτούνται περισσότερα από τέσσερα χρώματα για να χρωματιστούν οι περιοχές οποιουδήποτε χάρτη, έτσι ώστε καμία από τις δύο γειτονικές περιοχές να μην έχει το ίδιο χρώμα. Δίπλα σημαίνει ότι δύο περιοχές μοιράζονται ένα κοινό τμήμα καμπύλης ορίου και όχι απλώς μια γωνία όπου συναντώνται τρεις ή περισσότερες περιοχές [28]. Αρχικά, η απόδειξη αυτή δεν έγινε αποδεκτή από όλους τους μαθηματικούς, επειδή η απόδειξη με τη βοήθεια του υπολογιστή ήταν ανέφικτη για έναν άνθρωπο να την ελέγξει με το χέρι [29]. Η απόδειξη έχει κερδίσει ευρεία αποδοχή από τότε, αν και παραμένουν ορισμένοι αμφισβητίες [30].

Το θεώρημα των τεσσάρων χρωμάτων αποδείχθηκε το 1976 από τους Kenneth Appel και Wolfgang Haken [31] μετά από πολλές λανθασμένες αποδείξεις και αντιπαραδείγματα (σε αντίθεση με το θεώρημα των πέντε χρωμάτων, που αποδείχθηκε το 1800, το οποίο δηλώνει ότι πέντε χρώματα είναι αρκετά για να χρωματιστεί ένας χάρτης). Για να διαλυθούν τυχόν εναπομείνασες αμφιβολίες σχετικά με την απόδειξη των Appel - Haken, μια απλούστερη απόδειξη που χρησιμοποιεί τις ίδιες ιδέες και εξακολουθεί να βασίζεται σε υπολογιστές δημοσιεύθηκε το 1997 από τους Robertson, Sanders, Seymour και Thomas [32]. Το 2005, το θεώρημα αποδείχθηκε επίσης από τον Georges Gonthier με λογισμικό γενικής χρήσης για την επίλυση θεωρημάτων [28].



Εικόνα 3: Παράδειγμα χάρτη με 4-χρώματα

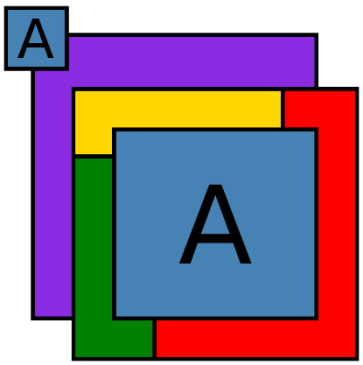
Ακριβής διατύπωση του θεωρήματος

Σε γραφηματοθεωρητικούς όρους, το θεώρημα δηλώνει ότι για επίπεδο γράφημα $\boldsymbol{G}$, ο χρωματικός αριθμός του είναι $\boldsymbol{\chi(G) \leq 4}$.

Η διαισθητική δήλωση του θεωρήματος των τεσσάρων χρωμάτων - "δεδομένου οποιουδήποτε διαχωρισμού ενός επιπέδου σε συνεχόμενες περιοχές, οι περιοχές μπορούν να χρωματιστούν χρησιμοποιώντας το πολύ τέσσερα χρώματα έτσι ώστε καμία από τις δύο γειτονικές περιοχές να μην έχει το ίδιο χρώμα" - πρέπει να ερμηνευτεί κατάλληλα για να είναι σωστή [33].

Πρώτον, οι περιοχές είναι γειτονικές εάν μοιράζονται ένα οριακό τμήμα- δύο περιοχές που μοιράζονται μόνο μεμονωμένα οριακά σημεία δεν θεωρούνται γειτονικές. (Διαφορετικά, ένας χάρτης σε σχήμα κυκλικού διαγράμματος θα καθιστούσε έναν αυθαίρετα μεγάλο αριθμό περιοχών "γειτονικές" μεταξύ τους σε μια κοινή γωνία και θα απαιτούσε αυθαίρετα μεγάλο αριθμό χρωμάτων ως αποτέλεσμα) [33].

Δεύτερον, δεν επιτρέπονται παράξενες περιοχές, όπως αυτές με πεπερασμένο εμβαδόν αλλά απείρως μεγάλη περίμετρο- χάρτες με τέτοιες περιοχές μπορεί να απαιτούν περισσότερα από τέσσερα χρώματα (Για να είμαστε ασφαλείς, μπορούμε να περιορίσουμε τις περιοχές των οποίων τα όρια αποτελούνται από πεπερασμένα πολλά ευθύγραμμα τμήματα. Επιτρέπεται μια

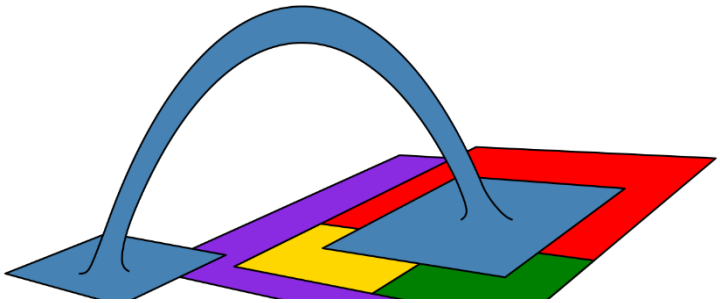


Εικόνα 4: Παράδειγμα με 5 χρώματα

περιοχή να έχει θύλακες, δηλαδή να περιβάλλει εξ' ολοκλήρου μια ή περισσότερες άλλες περιοχές). Σημειώστε ότι η έννοια της "συνεχούς περιοχής" (τεχνικά: συνδεδεμένο ανοικτό υποσύνολο του επιπέδου) δεν είναι ίδια με την έννοια της "χώρας" στους κανονικούς χάρτες, αφού οι χώρες δεν είναι απαραίτητο να είναι συνεχόμενες (μπορεί να έχουν θύλακες, π.χ. η επαρχία Cabinda ως τμήμα της Αγκόλας, το Ναχτσιβάν ως τμήμα του Αζερμπαϊτζάν, το Καλίνινγκραντ ως τμήμα της Ρωσίας, η Γαλλία με τα υπερπόντια εδάφη της και η Αλάσκα ως τμήμα των Ηνωμένων Πολιτειών δεν είναι συνεχόμενες). Εάν απαιτούσαμε να λάβει ολόκληρο το έδαφος μιας χώρας το ίδιο χρώμα, τότε τα τέσσερα χρώματα δεν είναι πάντα επαρκή. Για παράδειγμα, θεωρήστε έναν απλουστευμένο χάρτη, όπως στην Εικόνα 4 [33]:

Σε αυτόν τον χάρτη, οι δύο περιοχές με την ένδειξη Α ανήκουν στην ίδια χώρα. Αν θέλαμε αυτές οι περιοχές να λάβουν το ίδιο χρώμα, τότε θα απαιτούνταν πέντε χρώματα, αφού οι δύο περιοχές Α μαζί γειτνιάζουν με τέσσερις άλλες περιοχές, καθεμία από τις οποίες γειτνιάζει με όλες τις άλλες.

Το να αναγκάσετε δύο ξεχωριστές περιοχές να έχουν το ίδιο χρώμα μπορεί να μοντελοποιηθεί με την προσθήκη μιας "λαβής" που τις ενώνει εκτός του επιπέδου, όπως φαίνεται στην Εικόνα 5 [33]:



Εικόνα 5: Παράδειγμα με προσθήκη «λαβής»

Μια τέτοια κατασκευή καθιστά το πρόβλημα ισοδύναμο με τον χρωματισμό ενός χάρτη σε έναν τόρο (μια επιφάνεια γένους 1), ο οποίος απαιτεί έως και 7 χρώματα για έναν αυθαίρετο χάρτη. Μια παρόμοια κατασκευή εφαρμόζεται επίσης εάν ένα μόνο χρώμα χρησιμοποιείται για πολλές διαχωρισμένες περιοχές, όπως για τα υδάτινα σώματα σε πραγματικούς χάρτες, ή εάν υπάρχουν περισσότερες χώρες με διαχωρισμένες περιοχές. Σε τέτοιες περιπτώσεις μπορεί να απαιτούνται περισσότερα χρώματα με αυξανόμενο γένος της προκύπτουσας επιφάνειας [30].

Μια απλούστερη διατύπωση του θεωρήματος χρησιμοποιεί τη θεωρία γραφημάτων. Το σύνολο των περιοχών ενός χάρτη μπορεί να αναπαρασταθεί πιο αφηρημένα ως ένα μη κατευθυνόμενο γράφημα που έχει μια κορυφή για κάθε περιοχή και μια ακμή για κάθε ζεύγος περιοχών που μοιράζονται ένα οριακό τμήμα.

Το γράφημα αυτό είναι επίπεδο, μπορεί να σχεδιαστεί στο επίπεδο χωρίς διασταυρώσεις τοποθετώντας κάθε κορυφή σε μια αυθαίρετα επιλεγμένη θέση εντός της περιοχής στην οποία αντιστοιχεί και σχεδιάζοντας τις ακμές ως καμπύλες χωρίς διασταυρώσεις που οδηγούν από την κορυφή μιας περιοχής, μέσω ενός κοινού οριακού τμήματος, στην κορυφή μιας γειτονικής περιοχής. Αντίστροφα, κάθε επίπεδο γράφημα μπορεί να σχηματιστεί από ένα χάρτη με αυτόν τον τρόπο. Στη γραφοθεωρητική ορολογία, το θεώρημα των τεσσάρων χρωμάτων δηλώνει ότι οι κορυφές κάθε επίπεδου γραφήματος μπορούν να χρωματιστούν με το πολύ τέσσερα χρώματα, έτσι ώστε καμία από τις δύο γειτονικές κορυφές να μην λαμβάνει το ίδιο χρώμα, ή για συντομία [30]:

«Κάθε επίπεδη γραφική παράσταση είναι τετραχρωματιζόμενη»

Απόδειξη με υπολογιστή

Κατά τη διάρκεια των δεκαετιών του 1960 και 1970, ο Γερμανός μαθηματικός Heinrich Heesch ανέπτυξε μεθόδους χρήσης υπολογιστών για την αναζήτηση μιας απόδειξης. Ειδικότερα, ήταν ο πρώτος που χρησιμοποίησε την αποφόρτιση για την απόδειξη του θεωρήματος, η οποία αποδείχθηκε σημαντική στο σκέλος του αναπόφευκτου της επακόλουθης απόδειξης των Appel-Haken. Επίσης επέκτεινε την έννοια της αναγωγιμότητας και, μαζί με τον Ken Durre, ανέπτυξε ένα τεστ υπολογιστών γι' αυτήν. Δυστυχώς, σε αυτή την κρίσιμη συγκυρία, δεν μπόρεσε να εξασφαλίσει τον απαραίτητο χρόνο σε υπερυπολογιστή για να συνεχίσει το έργο του.

Άλλοι υιοθέτησαν τις μεθόδους του, συμπεριλαμβανομένης της προσέγγισής του με τη βοήθεια υπολογιστή, ενώ άλλες ομάδες μαθηματικών έκαναν αγώνα δρόμου για να ολοκληρώσουν τις αποδείξεις, ο Kenneth Appel και ο Wolfgang Haken στο Πανεπιστήμιο του Ιλινόις ανακοίνωσαν, στις 21 Ιουνίου 1976, ότι είχαν αποδείξει το θεώρημα, για το οποίο βοήθησε σε κάποιες αλγοριθμικές εργασίες ο John A. Koch.

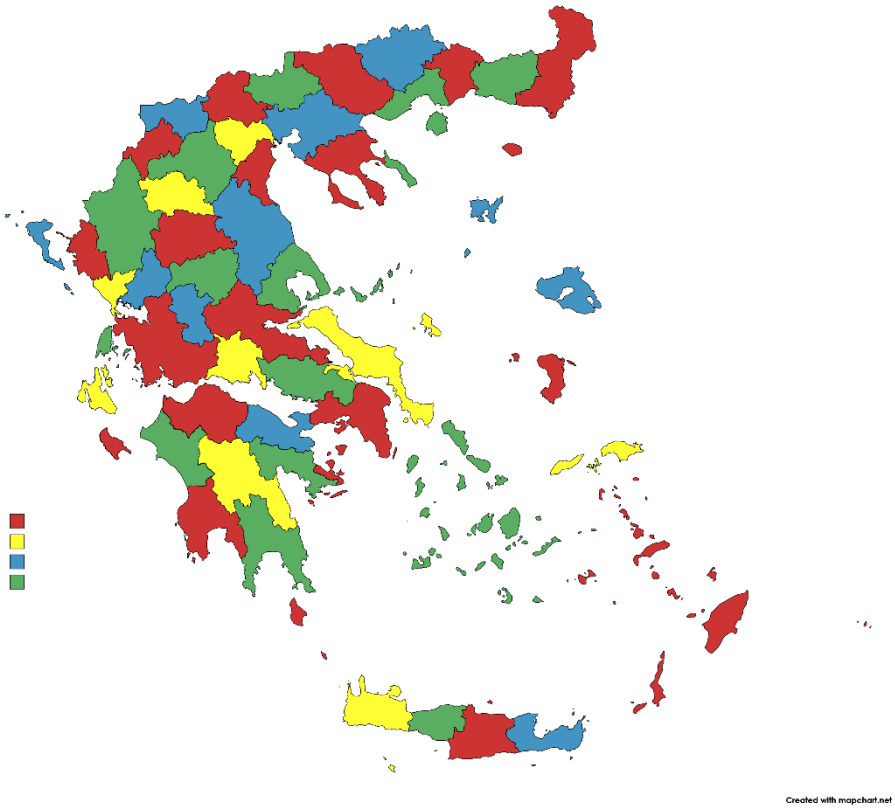
Αν η εικασία των τεσσάρων χρωμάτων ήταν ψευδής, θα υπήρχε τουλάχιστον ένας χάρτης με τον μικρότερο δυνατό αριθμό περιοχών που απαιτεί πέντε χρώματα. Η απόδειξη έδειξε ότι ένα τέτοιο ελάχιστο αντιπαράδειγμα δεν μπορεί να υπάρξει, μέσω της χρήσης δύο τεχνικών εννοιών [30]:

1. Ένα αναπόφευκτο σύνολο είναι ένα σύνολο από διαμορφώσεις τέτοιες ώστε κάθε χάρτης που ικανοποιεί κάποιες αναγκαίες συνθήκες για να είναι ένας ελάχιστος μη-4-χρωματιζόμενος τριγωνισμός (όπως το να έχει ελάχιστο βαθμό 5) πρέπει να έχει τουλάχιστον μία διαμόρφωση από αυτό το σύνολο.

2. Μια αναγώγιμη διαμόρφωση είναι μια διάταξη χωρών που δεν μπορεί να εμφανιστεί σε ένα ελάχιστο αντιπαράδειγμα. Εάν ένας χάρτης περιέχει μια αναγώγιμη διαμόρφωση, ο χάρτης μπορεί να αναχθεί σε έναν μικρότερο χάρτη. Αυτός ο μικρότερος χάρτης έχει τη συνθήκη ότι αν μπορεί να χρωματιστεί με τέσσερα χρώματα, αυτό ισχύει και για τον αρχικό χάρτη. Αυτό σημαίνει ότι αν ο αρχικός χάρτης δεν μπορεί να χρωματιστεί με τέσσερα χρώματα, ούτε ο μικρότερος χάρτης μπορεί να χρωματιστεί και έτσι ο αρχικός χάρτης δεν είναι ελάχιστος.

Χρησιμοποιώντας μαθηματικούς κανόνες και διαδικασίες βασισμένες στις ιδιότητες των αναγώγιμων διαμορφώσεων, οι Appel και Haken βρήκαν ένα αναπόφευκτο σύνολο αναγώγιμων διαμορφώσεων, αποδεικνύοντας έτσι ότι ένα ελάχιστο αντιπαράδειγμα στην εικασία των τεσσάρων χρωμάτων δεν μπορούσε να υπάρξει. Η απόδειξή τους μείωσε το άπειρο των πιθανών χαρτών σε 1.834 αναγώγιμες διαμορφώσεις (που αργότερα μειώθηκαν σε 1.482), οι οποίες έπρεπε να ελεγχθούν μία προς μία από υπολογιστή και χρειάστηκαν πάνω από χίλιες ώρες. Αυτό το μέρος της εργασίας που αφορά την αναγωγιμότητα ελέγχθηκε ανεξάρτητα διπλά με διαφορετικά προγράμματα και υπολογιστές. Ωστόσο, το μέρος της απόδειξης που αφορά την αναπόφευκτη διαμόρφωση επαληθεύτηκε σε πάνω από 400 σελίδες μικροφωτογραφιών, οι οποίες έπρεπε να ελεγχθούν με το χέρι με τη βοήθεια της κόρης του Haken, Dorothea Blostein [34].

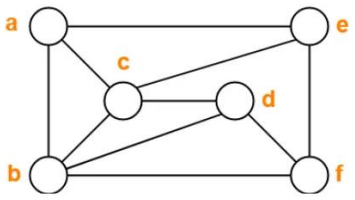
Η ανακοίνωση των Appel και Haken αναφέρθηκε ευρέως από τα ειδησεογραφικά μέσα ενημέρωσης σε όλο τον κόσμο και το τμήμα μαθηματικών του Πανεπιστημίου του Ιλινόις χρησιμοποίησε μια σφραγίδα ταχυδρομείου που ανέφερε "Τέσσερα χρώματα αρκούν". Ταυτόχρονα η ασυνήθιστη φύση της απόδειξης - ήταν το πρώτο σημαντικό θεώρημα που αποδείχθηκε με εκτεταμένη βοήθεια υπολογιστή - και η πολυπλοκότητα του επαληθεύσιμου τμήματος από τον άνθρωπο προκάλεσε σημαντική διαμάχη [30].



Εικόνα 6: Χάρτης της Ελλάδος χρωματισμένος με 4 χρώματα

Παράδειγμα αλγορίθμου σε Python

Ας πούμε ότι έχουμε ένα γράφημα όπως στην διπλανή Εικόνα 7 και το πρόβλημα είναι ότι πρέπει να χρωματίσουμε κάθε κόμβο με διαφορετικό χρώμα για κάθε γειτονικό κόμβο.



Εικόνα 7: Χρωματισμός κόμβων

Πώς λύνει ο "άπληστος" αλγόριθμος χρωματισμού το πρόβλημα; Εδώ είναι ο αλγόριθμος αυτός:

1. Ξεκινήστε όλους τους κόμβους.

2. Ορίστε τον κόμβο για τον πρώτο χρωματισμό, η προτεραιότητα είναι ο κόμβος με τον μεγαλύτερο βαθμό.

3. Επιλέξτε τον υποψήφιο χρωματισμό με τη συνάρτηση χρώματος επιλογής χωρίς να υπάρχει γειτονικός κόμβος που να έχει το ίδιο χρώμα.

4. Ελέγξτε την επιλεξιμότητα του χρώματος, εάν είναι σε θέση να αποθηκευτεί στο σύνολο λύσεων.

5. Είναι πλήρης η λύση; Πηγαίνετε στο βήμα 2, αν όχι ακόμα.

Επειδή θέλουμε να λύσουμε το πρόβλημα με την Python, πρέπει να αναπαραστήσουμε το γράφημα με τον παρακείμενο πίνακα. Στη συνέχεια, θα ξεκινήσουμε το όνομα του κόμβου με γράμματα και έπειτα θα μετρήσουμε το βαθμό και θα καθορίσουμε το πιθανό χρώμα.

Παρακάτω είναι το παράδειγμα του αλγορίθμου:

```
# Adjacent Matrix
G = [[ 0, 1, 1, 0, 1, 0],
    [ 1, 0, 1, 1, 0, 1],
    [ 1, 1, 0, 1, 1, 0],
    [ 0, 1, 1, 0, 0, 1],
    [ 1, 0, 1, 0, 0, 1],
    [ 0, 1, 0, 1, 1, 0]]

# inisiate the name of node.
node = "abcdef"
t_={}
for i in range(len(G)):
  t_[node[i]] = i
# count degree of all node.
degree =[]
for i in range(len(G)):
  degree.append(sum(G[i]))
# inisiate the posible color
colorDict = {}
for i in range(len(G)):
  colorDict[node[i]]=["Blue","Red","Yellow","Green"]

# sort the node depends on the degree
sortedNode=[]
indeks = []
```

```
# use selection sort
for i in range(len(degree)):
  _max = 0
  j = 0
  for j in range(len(degree)):
    if j not in indeks:
      if degree[j] > _max:
        _max = degree[j]
        idx = j
  indeks.append(idx)
  sortedNode.append(node[idx])

# The main process
theSolution={}
for n in sortedNode:
  setTheColor = colorDict[n]
  theSolution[n] = setTheColor[0]
  adjacentNode = G[t_[n]]
  for j in range(len(adjacentNode)):
    if adjacentNode[j]==1 and (setTheColor[0] in colorDict[node[j]]):
      colorDict[node[j]].remove(setTheColor[0])

# Print the solution
for t,w in sorted(theSolution.items()):
  print("Node",t," = ",w)

# The main process
theSolution={}
for n in sortedNode:
  setTheColor = colorDict[n]
  theSolution[n] = setTheColor[0]
  adjacentNode = G[t_[n]]
  for j in range(len(adjacentNode)):
    if adjacentNode[j]==1 and (setTheColor[0] in colorDict[node[j]]):
      colorDict[node[j]].remove(setTheColor[0])

# Print the solution
for t,w in sorted(theSolution.items()):
  print("Node",t," = ",w)
```

2.4 Ανάλυση του όρου Προγραμματισμός με Περιορισμούς (CP)

Ο προγραμματισμός περιορισμών (CP) [35] είναι ένα παράδειγμα για την επίλυση συνδυαστικών προβλημάτων που βασίζεται σε ένα ευρύ φάσμα τεχνικών από την τεχνητή νοημοσύνη, την επιστήμη των υπολογιστών και την επιχειρησιακή έρευνα. Στον προγραμματισμό με περιορισμούς, οι χρήστες δηλώνουν ρητά τους περιορισμούς στις εφικτές λύσεις για ένα σύνολο μεταβλητών απόφασης. Οι περιορισμοί διαφέρουν από τα κοινά πρωτεύοντα στοιχεία των προστακτικών γλωσσών προγραμματισμού στο ότι δεν καθορίζουν ένα βήμα ή μια ακολουθία βημάτων προς εκτέλεση, αλλά τις ιδιότητες μιας λύσης που πρέπει να βρεθεί. Εκτός από τους περιορισμούς, οι χρήστες πρέπει επίσης να καθορίσουν μια μέθοδο για την επίλυση αυτών των περιορισμών. Αυτό συνήθως βασίζεται σε τυποποιημένες μεθόδους

όπως η χρονολογική οπισθοδρόμηση και η διάδοση περιορισμών, αλλά μπορεί να χρησιμοποιεί προσαρμοσμένο κώδικα όπως μια ευρετική διακλάδωση συγκεκριμένου προβλήματος.

Ο προγραμματισμός με περιορισμούς έχει τη ρίζα του και μπορεί να εκφραστεί με τη μορφή του λογικού προγραμματισμού με περιορισμούς, ο οποίος ενσωματώνει περιορισμούς σε ένα λογικό πρόγραμμα. Αυτή η παραλλαγή του λογικού προγραμματισμού οφείλεται στους Jaffar και Lassez, οι οποίοι επέκτειναν το 1987 μια συγκεκριμένη κατηγορία περιορισμών που είχαν εισαχθεί στην Prolog II. Οι πρώτες υλοποιήσεις του λογικού προγραμματισμού με περιορισμούς ήταν η Prolog III, η CLP(R) και η CHIP [36].

Αντί του λογικού προγραμματισμού, οι περιορισμοί μπορούν να αναμειχθούν με τον λειτουργικό προγραμματισμό, την επανεγγραφή όρων και τις προστακτικές γλώσσες. Οι γλώσσες προγραμματισμού με ενσωματωμένη υποστήριξη για περιορισμούς περιλαμβάνουν την Oz (λειτουργικός προγραμματισμός) και το Kaleidoscope (προστακτικός προγραμματισμός). Ως επί το πλείστον, οι περιορισμοί υλοποιούνται σε προστακτικές γλώσσες μέσω εργαλειοθηκών επίλυσης περιορισμών, οι οποίες είναι ξεχωριστές βιβλιοθήκες για μια υπάρχουσα προστακτική γλώσσα.

2.4.1 Λογικός προγραμματισμός με περιορισμούς

Ο προγραμματισμός περιορισμών είναι μια ενσωμάτωση περιορισμών σε μια γλώσσα υποδοχής. Οι πρώτες γλώσσες υποδοχής που χρησιμοποιήθηκαν ήταν γλώσσες λογικού προγραμματισμού, οπότε ο τομέας αρχικά ονομάστηκε λογικός προγραμματισμός περιορισμών. Τα δύο παραδείγματα μοιράζονται πολλά σημαντικά χαρακτηριστικά, όπως οι λογικές μεταβλητές και η οπισθοδρόμηση. Σήμερα οι περισσότερες υλοποιήσεις της Prolog περιλαμβάνουν μία ή περισσότερες βιβλιοθήκες για λογικό προγραμματισμό με περιορισμούς.

Η διαφορά μεταξύ τους έγκειται κυρίως στο στυλ και την προσέγγισή τους στη μοντελοποίηση του κόσμου. Ορισμένα προβλήματα είναι πιο φυσικό (και συνεπώς απλούστερο) να γράφονται ως λογικά προγράμματα, ενώ ορισμένα είναι πιο φυσικό να γράφονται ως προγράμματα περιορισμών.

Η προσέγγιση του προγραμματισμού περιορισμών συνίσταται στην αναζήτηση μιας κατάστασης του περιβάλλοντος στην οποία ικανοποιείται ταυτόχρονα ένας μεγάλος αριθμός περιορισμών. Ένα πρόβλημα δηλώνεται συνήθως ως μια κατάσταση του περιβάλλοντος που περιέχει έναν αριθμό άγνωστων μεταβλητών. Το πρόγραμμα περιορισμών αναζητά τιμές για όλες τις μεταβλητές.

Ο χρονικός ταυτόχρονος προγραμματισμός περιορισμών (TCC) και ο μη ντετερμινιστικός χρονικός ταυτόχρονος προγραμματισμός περιορισμών (MJV) είναι παραλλαγές του προγραμματισμού περιορισμών που μπορούν να αντιμετωπίσουν το χρόνο.

2.4.2 Μοντέλα διαταραχής έναντι μοντέλων βελτίωσης

Οι γλώσσες για προγραμματισμό με περιορισμούς ακολουθούν μία από τις εξής δύο προσεγγίσεις [37]:

1. Μοντέλο βελτίωσης: οι μεταβλητές του προβλήματος είναι αρχικά μη καθορισμένες και κάθε μεταβλητή θεωρείται ότι μπορεί να περιέχει οποιαδήποτε τιμή περιλαμβάνεται στο εύρος ή το πεδίο εφαρμογής της. Καθώς ο υπολογισμός εξελίσσεται, οι τιμές στο πεδίο μιας μεταβλητής περικόπτονται εάν αποδειχθεί ότι είναι ασύμβατες με τις πιθανές τιμές άλλων μεταβλητών, μέχρι να βρεθεί μια μοναδική τιμή για κάθε μεταβλητή.

2. Μοντέλο διαταραχής: στις μεταβλητές του προβλήματος αποδίδεται μία μόνο αρχική τιμή. Σε διαφορετικές χρονικές στιγμές μία ή περισσότερες μεταβλητές δέχονται διαταραχές (αλλαγές στην παλιά τους τιμή) και το σύστημα διαδίδει την αλλαγή προσπαθώντας να αναθέσει νέες τιμές σε άλλες μεταβλητές που είναι συνεπείς με τη διαταραχή.

Η διάδοση περιορισμών σε προβλήματα ικανοποίησης περιορισμών είναι ένα τυπικό παράδειγμα μοντέλου βελτίωσης, ενώ τα λογιστικά φύλλα είναι ένα τυπικό παράδειγμα μοντέλου διαταραχών.

Το μοντέλο βελτίωσης είναι πιο γενικό, καθώς δεν περιορίζει τις μεταβλητές να έχουν μία μόνο τιμή, μπορεί να οδηγήσει σε πολλές λύσεις του ίδιου προβλήματος. Ωστόσο, το μοντέλο διαταραχής είναι πιο διαισθητικό για τους προγραμματιστές που χρησιμοποιούν μικτές γλώσσες αντικειμενοστρεφούς προγραμματισμού με επιτακτικούς περιορισμούς [38].

2.4.3 Τομείς

Οι περιορισμοί που χρησιμοποιούνται στον προγραμματισμό με περιορισμούς είναι συνήθως σε ορισμένους συγκεκριμένους τομείς. Μερικοί δημοφιλείς τομείς για τον προγραμματισμό με περιορισμούς είναι οι εξής [39]:

- boolean πεδία, όπου ισχύουν μόνο αληθείς/ψευδείς περιορισμοί (πρόβλημα SAT)
- ακέραιοι τομείς, ορθολογικοί τομείς
- τομείς διαστημάτων, ιδίως για προβλήματα χρονοπρογραμματισμού
- γραμμικά πεδία, όπου περιγράφονται και αναλύονται μόνο γραμμικές συναρτήσεις (αν και υπάρχουν προσεγγίσεις για μη γραμμικά προβλήματα)
- πεπερασμένα πεδία, όπου οι περιορισμοί ορίζονται σε πεπερασμένα σύνολα
- μικτά πεδία, που περιλαμβάνουν δύο ή περισσότερα από τα παραπάνω

Οι πεπερασμένοι τομείς είναι ένας από τους πιο επιτυχημένους τομείς του προγραμματισμού με περιορισμούς. Σε ορισμένους τομείς (όπως η επιχειρησιακή έρευνα) ο προγραμματισμός περιορισμών ταυτίζεται συχνά με τον προγραμματισμό περιορισμών σε πεπερασμένα πεδία.

2.4.4 Διάδοση περιορισμών

Οι τοπικές συνθήκες συνέπειας είναι ιδιότητες των προβλημάτων ικανοποίησης περιορισμών που σχετίζονται με τη συνέπεια υποσυνόλων μεταβλητών ή περιορισμών. Μπορούν να χρησιμοποιηθούν για τη μείωση του χώρου αναζήτησης και την ευκολότερη επίλυση του προβλήματος. Αξιοποιούνται διάφορα είδη συνθηκών τοπικής συνέπειας, συμπεριλαμβανομένης της συνέπειας κόμβων, της συνέπειας τόξων και της συνέπειας μονοπατιών [40].

Κάθε συνθήκη τοπικής συνέπειας μπορεί να επιβληθεί από έναν μετασχηματισμό που αλλάζει το πρόβλημα χωρίς να αλλάζει τις λύσεις του. Ένας τέτοιος μετασχηματισμός ονομάζεται διάδοση περιορισμών. Η διάδοση περιορισμών λειτουργεί με τη μείωση των περιοχών των μεταβλητών, την ενίσχυση των περιορισμών ή τη δημιουργία νέων. Αυτό οδηγεί σε μείωση του χώρου αναζήτησης, καθιστώντας το πρόβλημα ευκολότερο να επιλυθεί από ορισμένους αλγορίθμους. Η διάδοση περιορισμών μπορεί επίσης να χρησιμοποιηθεί ως ένας ελεγκτής μη ικανοποιησιμότητας, ελλιπής γενικά αλλά πλήρης σε ορισμένες ειδικές περιπτώσεις [40].

2.4.5 Επίλυση περιορισμών

Υπάρχουν τρεις κύριες αλγοριθμικές τεχνικές για την επίλυση προβλημάτων ικανοποίησης περιορισμών: αναζήτηση με οπισθοδρόμηση, τοπική αναζήτηση και δυναμικός προγραμματισμός [35].

Η **αναζήτηση με οπισθοδρόμηση** είναι ένας γενικός αλγόριθμος για την εύρεση όλων (ή ορισμένων) λύσεων σε ορισμένα υπολογιστικά προβλήματα, ιδίως σε προβλήματα ικανοποίησης περιορισμών, ο οποίος κατασκευάζει σταδιακά υποψήφιες λύσεις και εγκαταλείπει μια υποψήφια λύση ("οπισθοδρομεί") μόλις διαπιστώσει ότι η υποψήφια λύση δεν μπορεί ενδεχομένως να ολοκληρωθεί σε μια έγκυρη λύση.

Η **τοπική αναζήτηση** είναι μια ελλιπής μέθοδος για την εύρεση λύσης σε ένα πρόβλημα. Βασίζεται στην επαναληπτική βελτίωση μιας ανάθεσης των μεταβλητών μέχρι να ικανοποιηθούν όλοι οι περιορισμοί. Ειδικότερα, οι αλγόριθμοι τοπικής αναζήτησης συνήθως τροποποιούν την τιμή μιας μεταβλητής σε μια ανάθεση σε κάθε βήμα. Η νέα ανάθεση είναι κοντά στην προηγούμενη στο χώρο της ανάθεσης, εξ’ ου και η ονομασία τοπική αναζήτηση.

Ο **δυναμικός προγραμματισμός** είναι τόσο μια μαθηματική μέθοδος βελτιστοποίησης όσο και μια μέθοδος προγραμματισμού σε υπολογιστή. Αναφέρεται στην απλοποίηση ενός πολύπλοκου προβλήματος με τη διάσπασή του σε απλούστερα υποπροβλήματα με αναδρομικό τρόπο. Ενώ ορισμένα προβλήματα αποφάσεων δεν μπορούν να διαχωριστούν με αυτόν τον τρόπο, οι αποφάσεις που καλύπτουν πολλά χρονικά σημεία συχνά διαχωρίζονται αναδρομικά. Ομοίως, στην επιστήμη των υπολογιστών, εάν ένα πρόβλημα μπορεί να επιλυθεί βέλτιστα με τη διάσπασή του σε υποπροβλήματα και στη συνέχεια με αναδρομική εύρεση των βέλτιστων λύσεων στα υποπροβλήματα, τότε λέγεται ότι έχει βέλτιστη υποδομή.

Κεφάλαιο 3ο: Εισαγωγή στο λογισμικό Google OR – Tools

Το OR-Tools είναι ένα λογισμικό ανοικτού κώδικα για συνδυαστική βελτιστοποίηση, η οποία επιδιώκει να βρει την καλύτερη λύση σε ένα πρόβλημα από ένα πολύ μεγάλο σύνολο πιθανών λύσεων. Ακολουθούν ορισμένα παραδείγματα προβλημάτων που επιλύει το OR-Tools:

- **Δρομολόγηση οχημάτων:** Εύρεση βέλτιστων διαδρομών για στόλους οχημάτων που παραλαμβάνουν και παραδίδουν πακέτα λαμβάνοντας υπόψη περιορισμούς (π.χ. "αυτό το φορτηγό δεν μπορεί να μεταφέρει περισσότερα από 20.000 κιλά" ή "όλες οι παραδόσεις πρέπει να γίνουν μέσα σε ένα παράθυρο δύο ωρών").

- **Προγραμματισμός:** Εύρεση του βέλτιστου χρονοδιαγράμματος για ένα σύνθετο σύνολο εργασιών, ορισμένες από τις οποίες πρέπει να εκτελεστούν πριν από άλλες, σε ένα σταθερό σύνολο μηχανημάτων ή άλλων πόρων.

- **Πακετάρισμα καλαθιού:** Πακετάρισμα όσο το δυνατόν περισσότερων αντικειμένων διαφόρων μεγεθών σε σταθερό αριθμό καλαθιών με μέγιστη χωρητικότητα.

Στις περισσότερες περιπτώσεις, προβλήματα όπως αυτά έχουν έναν τεράστιο αριθμό πιθανών λύσεων - πάρα πολλές για να τις αναζητήσει όλες ένας υπολογιστής. Για να το ξεπεράσει αυτό, το OR-Tools χρησιμοποιεί αλγορίθμους τελευταίας τεχνολογίας για να περιορίσει το σύνολο αναζήτησης, προκειμένου να βρει μια βέλτιστη (ή σχεδόν βέλτιστη) λύση.

Το OR-Tools περιλαμβάνει λύτες για:

- **Προγραμματισμό με περιορισμούς (CP)**

  Ένα σύνολο τεχνικών για την εύρεση εφικτών λύσεων σε ένα πρόβλημα που εκφράζεται ως περιορισμοί (π.χ. μια αίθουσα δεν μπορεί να χρησιμοποιηθεί για δύο εκδηλώσεις ταυτόχρονα, ή η απόσταση από τις καλλιέργειες πρέπει να είναι μικρότερη από το μήκος του σωλήνα, ή δεν μπορούν να εγγραφούν περισσότερες από πέντε τηλεοπτικές εκπομπές ταυτόχρονα).

- **Γραμμικός και μικτός-ολοκληρωτικός προγραμματισμός**

  Ο γραμμικός βελτιστοποιητής Glop βρίσκει τη βέλτιστη τιμή μιας γραμμικής αντικειμενικής συνάρτησης, δεδομένου ενός συνόλου γραμμικών ανισοτήτων ως περιορισμών (π.χ. ανάθεση ατόμων σε θέσεις εργασίας ή εύρεση της καλύτερης κατανομής ενός συνόλου πόρων με ταυτόχρονη ελαχιστοποίηση του κόστους). Το Glop και το λογισμικό μικτού ακέραιου προγραμματισμού SCIP είναι επίσης διαθέσιμα μέσω της υπηρεσίας Google Apps Script Optimization Service.

- **Δρομολόγηση οχημάτων**

  Μια εξειδικευμένη βιβλιοθήκη για τον εντοπισμό των καλύτερων διαδρομών οχημάτων δεδομένων περιορισμών.

- **Αλγόριθμοι γραφημάτων**

  Κώδικας για την εύρεση συντομότερων διαδρομών σε γραφήματα, ροές ελάχιστου κόστους, μέγιστες ροές και αναθέσεις γραμμικού αθροίσματος.

Συγκεκριμένα το OR-Tools περιέχει δύο λογισμικά που επιλύουν προβλήματα προγραμματισμού με περιορισμούς και είναι τα εξής:

1. Ο CP-SAT Solver
2. Ο original CP Solver

Ο CP-SAT Solver είναι τεχνολογικά ανώτερος από τον δεύτερο επιλυτή οπότε είναι μια καλύτερη επιλογή για την επίλυση του προβλήματος του σχολικού προγραμματισμού και θα προτιμηθεί στην συγκεκριμένη εργασία.

Κεφάλαιο 4ο: Ανάλυση του προβλήματος

4.1 Μαθηματική μοντελοποίηση

Στην παρούσα ενότητα θα γίνει αναπαράσταση του προβλήματα με τη χρήση μαθηματικής μοντελοποίησης. Ο συμβολισμός και οι παράμετροι που θα χρησιμοποιηθούν προκύπτουν παρακάτω:

- E – Το σύνολο των υπαρχόντων επιπέδων
- T – Το σύνολο των υπαρχόντων τμημάτων
- A – Το σύνολο των υπαρχόντων αντικειμένων
- $Kyklos\_Mathimatwn[e][a][w]$ – Ο Κύκλος Μαθημάτων ορίζει ανάλογα με το επίπεδο ($e \epsilon E$) και το αντικείμενο ($\alpha \epsilon A$) πόσες ώρες διδασκαλίας (w) αναλογούν εβδομαδιαίως.
- $Kathigites[o][a][w]$ – Το σύνολο των Καθηγητών περιέχει τα ονόματα (o) των καθηγητών, τα αντικείμενα ($\alpha \epsilon A$) τα οποία διδάσκει ο καθένας και τις μέγιστες ώρες (w) που μπορεί να διδάξει ο καθένας.
- $Xronikes\_Periodoi[m][p][d]$ – Το σύνολο των Χρονικών Περιόδων περιέχει τις ημέρες (m) με τις αντίστοιχες περιόδους (p) και την διάρκεια που αντιστοιχεί στην καθεμία (d).

Οι περιορισμοί που ορίστηκαν είναι οι εξής:

1. $\forall e \epsilon E, \forall t \epsilon T, \forall a \epsilon A, \forall w \epsilon Kyklos\_Mathimatwn$ έτσι ώστε

$$\sum \Omega\rho\acute{\omega}\nu[e][t][a] = w[e][a]$$

δηλαδή το σύνολο των ωρών κάθε τμήματος να συμπίπτει με τις ώρες που ορίζει ο Κύκλος Μαθημάτων.

2. $\forall e \epsilon E, \forall t \epsilon T, \forall p \epsilon Xronikes\_Periodoi$ έτσι ώστε

$$\sum_{0}^{p} \Pi\epsilon\rho\iota\acute{o}\delta\omega\nu[e][t][p] \leq 1$$

δηλαδή κάθε τμήμα μπορεί να κάνει μόνο ένα μάθημα σε κάθε χρονική περίοδο.

3. $\forall k \epsilon Kathigites, \forall p \epsilon Xronikes\_Periodoi, \forall e \epsilon E, \forall t \epsilon T, \forall a \epsilon A$ έτσι ώστε

$$\sum_{0}^{p} Περιόδων[e][t][α][k][p] \leq 1$$

δηλαδή ο κάθε καθηγητής να διδάσκει το πολύ σε ένα τμήμα τη φορά.

4. $\forall w, k \in Kathigites, \forall p \in Xronikes\_Periodoi, \forall e \in E, \forall t \in T, \forall a \in A$ έτσι ώστε

$$\sum_{0}^{k} Ωρών[e][t][α][k][p] \leq Max[k][w]$$

δηλαδή οι ώρες διδασκαλίας ενός καθηγητή να μην ξεπερνούν τις μέγιστες ώρες που προσδιορίζονται για αυτόν.

5. $\forall e \in E, \forall t \in T, \forall a \in A, \forall k \in Kathigites$ ορίζουμε μια νέα μεταβλητή ώστε $mathima[e][t][a][k] = 1$

Το μοντέλο από το οποίο προκύπτει το τελικό αποτέλεσμα είναι το εξής:

$$MaxZ = \sum_{0}^{e}\sum_{0}^{t}\sum_{0}^{a} Kathigites[e][t][a] * Xronikes\_Periodoi\,[e][t][a]$$

4.2 Ανάλυση της εφαρμογής

Το πρόγραμμα που δημιουργήθηκε μπορεί να λειτουργήσει σε Windows και Linux. Παρακάτω θα παρουσιαστεί το περιβάλλον στο οποίο αναπτύχθηκε, τα μέρη του κώδικα και πως λειτουργεί.

4.2.1 Οδηγίες για την εγκατάσταση του PyCharm

Το παρόν πρόγραμμα αναπτύχθηκε με την χρήση του PyCharm και για να λειτουργήσει θα πρέπει να είναι εγκατεστημένο στον υπολογιστή ή κάποιο παρόμοιο πρόγραμμα, με δυνατότητα ανάκτησης της βιβλιοθήκης OR Tools.

Το PyCharm είναι ένα διαπλατφορμικό IDE[5] που παρέχει συνεπή εμπειρία στα λειτουργικά συστήματα Windows, macOS και Linux, ενώ είναι διαθέσιμο σε δύο εκδόσεις: Professional και Community. Η έκδοση Community είναι ένα έργο ανοιχτού κώδικα και είναι δωρεάν, αλλά έχει λιγότερες δυνατότητες. Η έκδοση Professional είναι εμπορική και παρέχει ένα εξαιρετικό σύνολο εργαλείων και χαρακτηριστικών και είναι αυτή η οποία χρησιμοποιήθηκε [41].

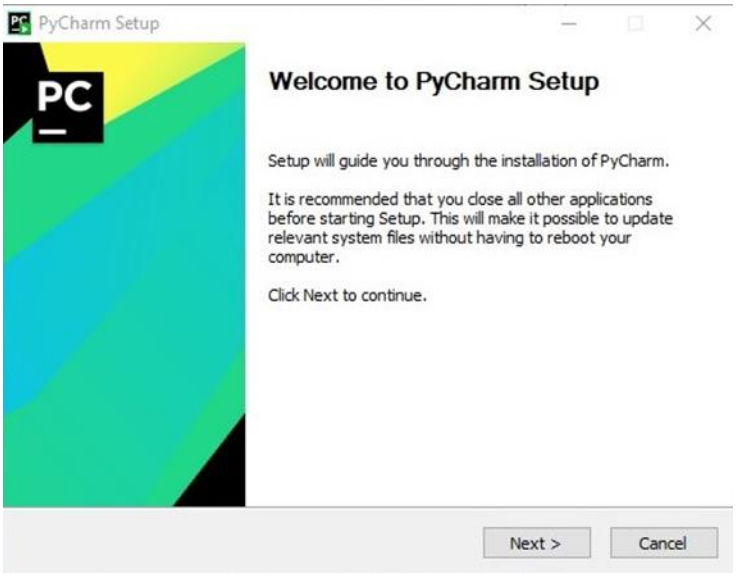
Βήματα εγκατάστασης PyCharm:

[5] Integrated Development Environment (IDE): Ένα ενοποιημένο περιβάλλον προγραμματισμού (IDE) είναι μια εφαρμογή λογισμικού που βοηθά τους προγραμματιστές να αναπτύσσουν αποτελεσματικά κώδικα λογισμικού. Αυξάνει την παραγωγικότητα των προγραμματιστών συνδυάζοντας δυνατότητες όπως η επεξεργασία, η κατασκευή, η δοκιμή και η συσκευασία λογισμικού σε μια εύχρηστη εφαρμογή.

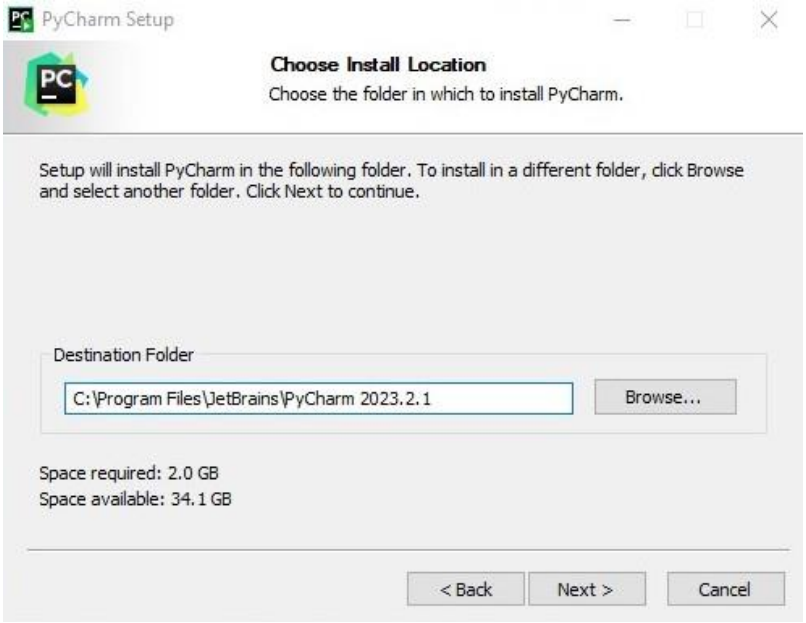
1. Επισκεπτόμενοι τη σελίδα

   https://www.jetbrains.com/pycharm/download/?section=windows

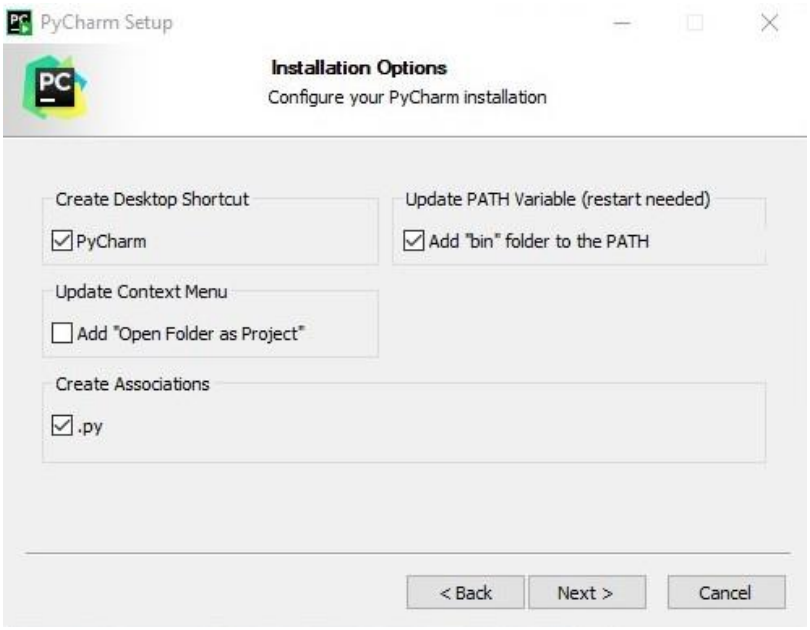
   κάνουμε λήψη του αρχείου ***pycharm-professional-2023.2.1.exe***
2. Εκτελούμε το αρχείο που κάναμε λήψη για να ξεκινήσει η εγκατάσταση.
3. Στο παράθυρο που εμφανίζεται κάνουμε κλικ στο κουμπί “Next”.



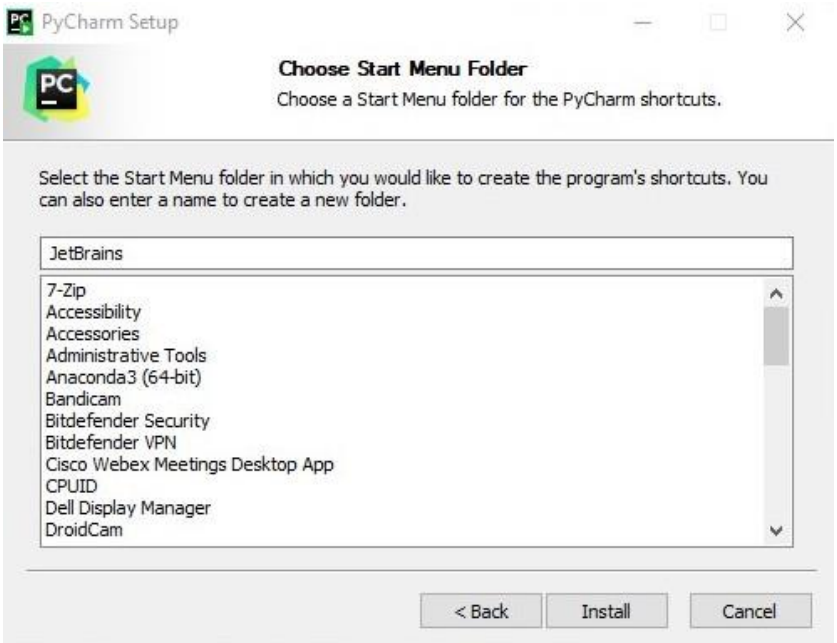
4. Στο επόμενο παράθυρο επιλέγουμε τον Φάκελο εγκατάστασης και κάνουμε κλικ στο κουμπί “Next”.



5. Στο επόμενο παράθυρο επιλέγουμε τα κουτάκια όπως παρουσιάζονται στην εικόνα παρακάτω και κάνουμε κλικ στο κουμπί “Next”.



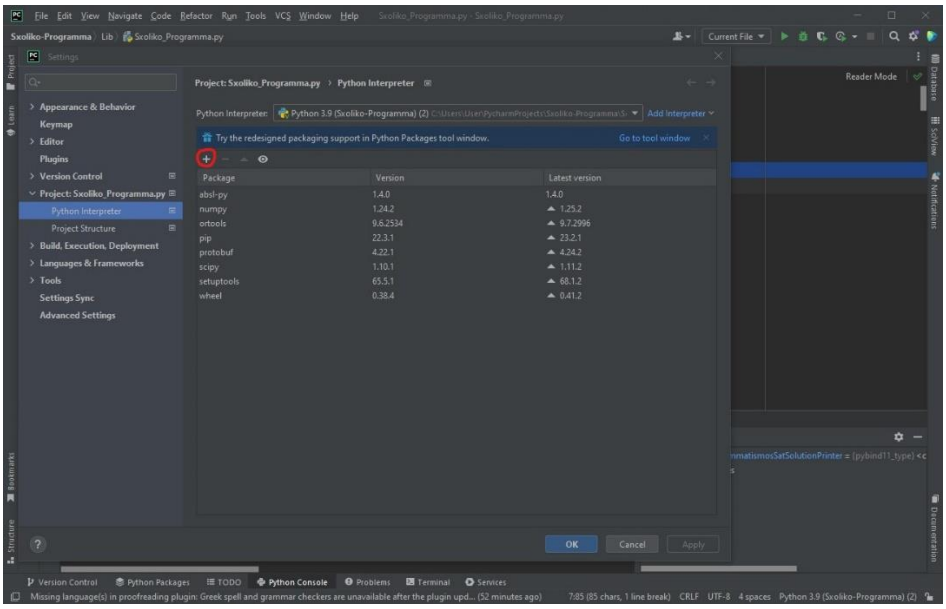
6. Στη συνέχεια κάνουμε κλικ στο κουμπί “Install” και περιμένουμε να ολοκληρωθεί η εγκατάσταση.



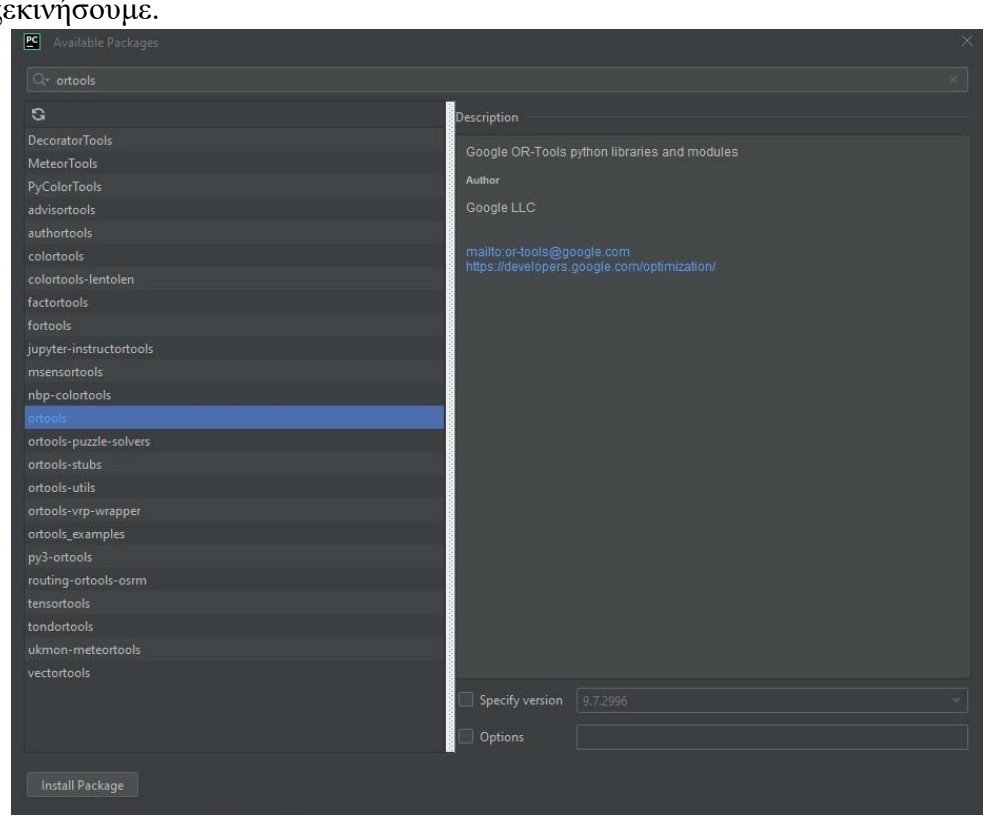
7. Τέλος επιλέγουμε το “Reboot now” και κάνουμε κλικ στο κουμπί “Finish”.

8. Το επόμενο βήμα είναι να ενεργοποιήσουμε το OR Tools ώστε να μπορούμε να το καλέσουμε στο πρόγραμμα. Άρα αφού εκτελέσουμε το PyCharm ακολουθούμε την εξής διαδρομή:

File→Settings→Project→Python Interpreter→+



9. Έπειτα αναζητούμε το ortools και κάνουμε “Install Package” και είμαστε έτοιμοι να ξεκινήσουμε.



4.2.2 Επίλυση του προβλήματος

Για να αναπαραστήσουμε το σχολικό πρόγραμμα πρέπει να δημιουργήσουμε μια κλάση όπου θα αποθηκεύονται όλα τα στοιχεία, όπως το επίπεδο, το τμήμα, το αντικείμενο, ο κύκλος μαθημάτων, ο καθηγητής, η ειδικότητα και η χρονική περίοδος.

```
class SxolikosProgrammatismosProblem():
    '''Δεδομένα του προβλήματος.'''
```

```
  def __init__(self, epipeda, tmimata, antikeimena, kyklos_mathimatwn, kathigites,
               eidikotites, xronikes_periodoi):
      self._epipeda = epipeda
      self._tmimata = tmimata
      self._antikeimena = antikeimena
      self._kyklos_mathimatwn = kyklos_mathimatwn
      assert len(self._kyklos_mathimatwn) == len(self._epipeda) * len(
          self._antikeimena), 'Κάποιος κύκλος μαθημάτων λείπει'
      for (epipedo, eidikotita) in self._kyklos_mathimatwn.keys():
          assert epipedo in self._epipeda, f'{epipedo} δεν υπάρχει στη μεταβλητή EPIPEDA'
          assert eidikotita in self._antikeimena, f'{eidikotita} δεν υπάρχει στη μεταβλητή EIDIKOTITA'

      self._kathigites = kathigites
      self._eidikotites = eidikotites
      assert len(self._eidikotites) == len(
          self._antikeimena), 'Λείπουν κάποιες γραμμές'
      for s, ts in self._eidikotites.items():
          assert s in self._antikeimena, f'{s} δεν υπάρχει στη μεταβλητή ANTIKEIMENA'
          for t in ts:
              assert t in self._kathigites, f'{t} δεν υπάρχει στη μεταβλητή KATHIGITES'

      self._xronikes_periodoi = xronikes_periodoi
```

Για τη δημιουργία μοντέλου θα δημιουργήσουμε μια μεταβλητή λογικού τύπου (Boolean), ώστε να αναπαραστήσουμε τη μεταβλητή απόφασής μας η οποία έχει 5 διαφορετικές μεταβλητές: το επίπεδο, το τμήμα, το αντικείμενο, τον καθηγητή και την περίοδο. Αφού θα είναι μεταβλητή λογικού τύπου θα παίρνει δύο τιμές 0 (false) ή 1 (true) αν ο καθηγητής είναι διαθέσιμος να αναλάβει το συγκεκριμένο επίπεδο – τμήμα και αντικείμενο, στην συγκεκριμένη περίοδο.

```
class SxolikosProgrmammatismosSolver():
    '''Παράδειγμα Επίλυσης.'''

    def __init__(self, problem: SxolikosProgrammatismosProblem):
        # Πρόβλημα
        self._problem = problem

        # Υπηρεσίες
        arithmos_epipedwn = len(self._problem.epipeda)
        self._all_epipeda = range(arithmos_epipedwn)
        arithmos_tmimatwn = len(self._problem.tmimata)
        self._all_tmimata = range(arithmos_tmimatwn)
        arithmos_antikeimenwn = len(self._problem.antikeimena)
        self._all_antikeimena = range(arithmos_antikeimenwn)
        arithmos_kathigitwn = len(self._problem.kathigites)
        self._all_kathigites = range(arithmos_kathigitwn)
        arithmos_xronikwn_periodwn = len(self._problem.xronikes_periodoi)
        self._all_xronikes_periodoi = range(arithmos_xronikwn_periodwn)

        # Δημιουργία μοντέλου
        self._model = cp_model.CpModel()

        # Δημιουργία μεταβλητών
        self._assignment = {}
        for epipedo_id, epipedo in enumerate(self._problem.epipeda):
            for tmima_id, tmima in enumerate(self._problem.tmimata):
                for antikeimeno_id, antikeimeno in
enumerate(self._problem.antikeimena):
                    for kathigitis_id, kathigitis in
```

```
enumerate(self._problem.kathigites):
                        for periodos_id, periodos in
enumerate(self._problem.xronikes_periodoi):
                            k = (epipedo_id, tmima_id, antikeimeno_id,
kathigitis_id, periodos_id)
                            n = f'{epipedo}-{tmima} S:{antikeimeno}
T:{kathigitis} Slot:{periodos}'
                            # Εκτύπωση Ονόματος
                            if kathigitis in
self._problem.eidikotita_kathigitwn(antikeimeno):
                                self._assignment[k] =
self._model.NewBoolVar(n)
                            else:
                                n = 'NO DISP ' + n
                                self._assignment[k] =
self._model.NewIntVar(0, 0, n)
```

Προφανώς χρειάζεται να υλοποιηθούν και οι περιορισμοί ώστε να είναι λειτουργικό το μοντέλο μας. Στον πρώτο περιορισμό εξασφαλίζουμε ότι κάθε τμήμα πρέπει να έχει τον απαραίτητο αριθμό μαθημάτων που προσδιορίζεται από τον κύκλο μαθημάτων. Οπότε για κάθε επίπεδο – τμήμα εξισώνουμε τα αντικείμενα που έχουν οριστεί από το μοντέλο αν είναι ίσα με τα αντικείμενα που ορίζονται από τον κύκλο μαθημάτων.

```
# Κάθε Τμήμα πρέπει να έχει τον απαραίτητο αριθμό μαθημάτων που προσδιορίζεται από τον Κύκλο
Μαθημάτων
for epipedo_id, epipedo in enumerate(self._problem.epipeda):
    for tmima_id in self._all_tmimata:
        for antikeimeno_id, antikeimeno in enumerate(self._problem.antikeimena):
            apaitoumeni_diarkeia = self._problem.kyklos_mathimatwn[epipedo, antikeimeno]
            #print(f'Τάξη:{epipedo} Μάθημα:{antikeimeno} Διάρκεια:{apaitoumeni_diarkeia}h')
            self._model.Add(
                sum(self._assignment[epipedo_id, tmima_id, antikeimeno_id, kathigitis_id, periodos_id] *
                    int(self._problem.diarkeia_periodou(periodos_id) * 10)
                    for kathigitis_id in self._all_kathigites
                    for periodos_id in self._all_xronikes_periodoi) == int(apaitoumeni_diarkeia * 10))
```

Στον δεύτερο περιορισμό εξασφαλίζουμε ότι κάθε τμήμα μπορεί να κάνει μόνο ένα μάθημα σε κάθε χρονική περίοδο.

```
# Κάθε τμήμα μπορεί να κάνει μόνο ένα μάθημα σε κάθε χρονική περίοδο
for epipedo_id in self._all_epipeda:
    for tmima_id in self._all_tmimata:
        for periodos_id in self._all_xronikes_periodoi:
            self._model.Add(
                sum([
                    self._assignment[epipedo_id, tmima_id, antikeimeno_id, kathigitis_id, periodos_id]
                    for antikeimeno_id in self._all_antikeimena
                    for kathigitis_id in self._all_kathigites
                ]) <= 1)
```

Στον τρίτο περιορισμό εξασφαλίζουμε ότι ο κάθε καθηγητής μπορεί να διδάσκει το πολύ σε ένα τμήμα τη φορά.

```
# Ο Καθηγητής μπορεί να διδάσκει το πολύ σε ένα τμήμα τη φορά
for kathigitis_id in self._all_kathigites:
    for periodos_id in self._all_xronikes_periodoi:
        self._model.Add(
```

```
            sum([
                self._assignment[epipedo_id, tmima_id, antikeimeno_id, kathigitis_id, periodos_id]
                for epipedo_id in self._all_epipeda
                for tmima_id in self._all_tmimata
                for antikeimeno_id in self._all_antikeimena
            ]) <= 1)
```

Στον τέταρτο περιορισμό είναι απαραίτητο να εξασφαλίσουμε ότι ο κάθε καθηγητής δεν θα υπερβεί τις μέγιστες ώρες εργασίας του.

```
# Μέγιστες ώρες εργασίας κάθε καθηγητή
for kathigitis_id in self._all_kathigites:
    self._model.Add(
        sum([
            self._assignment[epipedo_id, tmima_id, antikeimeno_id, kathigitis_id, periodos_id] *
            int(self._problem.diarkeia_periodou(periodos_id) * 10)
            for epipedo_id in self._all_epipeda
            for tmima_id in self._all_tmimata
            for antikeimeno_id in self._all_antikeimena
            for periodos_id in self._all_xronikes_periodoi
        ]) <= int(self._problem.kathigitis_max_wres(kathigitis_id) * 10))
```

Ο πέμπτος και τελευταίος περιορισμός είναι να εξασφαλίσουμε ότι ο κάθε καθηγητής θα κάνει το ίδιο μάθημα σε όλα τα τμήματα που θα οριστεί. Δηλαδή αν ο «Τάδε» καθηγητής κάνει τα μαθηματικά στο Γ3, θα πρέπει να είναι ο ίδιος καθηγητής και τις 3 ώρες που είναι απαραίτητες.

```
# Ο Καθηγητής κάνει όλες τις τάξεις στο ίδιο μάθημα
mathima_kathigiti = {}
for epipedo_id, epipedo in enumerate(self._problem.epipeda):
    for tmima_id, tmima in enumerate(self._problem.tmimata):
        for antikeimeno_id, antikeimeno in enumerate(self._problem.antikeimena):
            for kathigitis_id, kathigitis in enumerate(self._problem.kathigites):
                n = f'{epipedo}-{tmima} Μάθημα:{antikeimeno} Καθηγητής:{kathigitis}'
                mathima_kathigiti[epipedo_id, tmima_id, antikeimeno_id, kathigitis_id] =
self._model.NewBoolVar(n)
                temp_array = [
                    self._assignment[epipedo_id, tmima_id, antikeimeno_id, kathigitis_id, periodos_id]
                    for periodos_id in self._all_xronikes_periodoi
                ]
                self._model.AddMaxEquality(
                    mathima_kathigiti[epipedo_id, tmima_id, antikeimeno_id, kathigitis_id], temp_array)
            self._model.Add(
                sum([mathima_kathigiti[epipedo_id, tmima_id, antikeimeno_id, kathigitis_id]
                     for kathigitis_id in self._all_kathigites
                     ]) == 1)
```

Επειδή το βασικό πρόβλημα είναι να δημιουργηθεί ένα σχολικό πρόγραμμα θα πρέπει να δημιουργηθούν συναρτήσεις που θα εκτυπώνουν το πρόγραμμα. Στην αρχή δημιουργούμε μια συνάρτηση εκτύπωσης για το πρόγραμμα του κάθε καθηγητή.

```
def print_programma_kathigiti(self, kathigitis_id):
    onoma_kathigiti = self._problem.kathigitis_onoma(kathigitis_id)
    print(f'Καθηγητής: {onoma_kathigiti}')
    synolikes_wres_ergasias = 0
    for periodos_id, periodos in enumerate(self._problem.xronikes_periodoi):
```

```
      for epipedo_id, epipedo in enumerate(self._problem.epipeda):
        for tmima_id, tmima in enumerate(self._problem.tmimata):
          for antikeimeno_id, antikeimeno in enumerate(self._problem.antikeimena):
            key = (epipedo_id, tmima_id, antikeimeno_id, kathigitis_id, periodos_id)
            if self._solver.BooleanValue(self._assignment[key]):
              synolikes_wres_ergasias += self._problem.diarkeia_periodou(periodos_id)
              print(f'{periodos}: Τμήμα:{epipedo}-{tmima} Μάθημα:{antikeimeno}')
  print(f'Συνολικές ώρες εργασίας: {synolikes_wres_ergasias}h')
```

Στη συνέχεια δημιουργούμε μια συνάρτηση εκτύπωσης για το πρόγραμμα του κάθε τμήματος.

```
def print_programma_tmimatos(self, epipedo_id, tmima_id):
  epipedo = self._problem.epipeda[epipedo_id]
  tmima = self._problem.tmimata[tmima_id]
  print(f'Τμήμα: {epipedo}-{tmima}')
  synolikes_wres_ergasias = {}
  for ant in self._problem.antikeimena:
    synolikes_wres_ergasias[ant] = 0
  for periodos_id, periodos in enumerate(self._problem.xronikes_periodoi):
    for kathigitis_id, kathigitis in enumerate(self._problem.kathigites):
      for antikeimeno_id, antikeimeno in enumerate(self._problem.antikeimena):
        key = (epipedo_id, tmima_id, antikeimeno_id, kathigitis_id, periodos_id)
        if self._solver.BooleanValue(self._assignment[key]):
          synolikes_wres_ergasias[antikeimeno] += self._problem.diarkeia_periodou(periodos_id)
          print(f'{periodos}: Μάθημα:{antikeimeno} Καθηγητής:{kathigitis}')
  for (antikeimeno, wres) in synolikes_wres_ergasias.items():
    print(f'Συνολικές ώρες εργασίας για {antikeimeno}: {wres}h')
```

Τέλος, δημιουργούμε μια συνάρτηση εκτύπωσης για το συνολικό πρόγραμμα του σχολείου.

```
def print_programma_sxoleiou(self):
    print('Σχολείο:')
    for periodos_id, periodos in
enumerate(self._problem.xronikes_periodoi):
        tmp = f'{periodos}:'
        for epipedo_id, epipedo in enumerate(self._problem.epipeda):
            for tmima_id, tmima in enumerate(self._problem.tmimata):
                for antikeimeno_id, antikeimeno in
enumerate(self._problem.antikeimena):
                    for kathigitis_id, kathigitis in
enumerate(self._problem.kathigites):
                        key = (epipedo_id, tmima_id, antikeimeno_id,
kathigitis_id, periodos_id)
                        if
self._solver.BooleanValue(self._assignment[key]):
                            tmp += f' {epipedo}-
{tmima}:({antikeimeno},{kathigitis})'
        print(tmp)
```

Η συνάρτηση “solve” που ακολουθεί ουσιαστικά είναι αυτή που ουσιαστικά καλείται για να γίνει η τελική εκτύπωση όλων των προγραμμάτων και αν η κατάσταση της λύσης του προβλήματος είναι η βέλτιστης ή απλώς επιλύσιμη.

```
def solve(self):
    print('Επίλυση...')
    # Create Solver
```

```
    self._solver = cp_model.CpSolver()

    solution_printer = SxolikosProgrammatismosSatSolutionPrinter()
    status = self._solver.Solve(self._model, solution_printer)
    print('Κατάσταση: ', self._solver.StatusName(status))

    if status == cp_model.OPTIMAL or status == cp_model.FEASIBLE:
        print('\n# Καθηγητές')
        for kathigitis_id in self._all_kathigites:
            self.print_programma_kathigiti(kathigitis_id)

        print('\n# Τμήματα ')
        for epipedo_id in self._all_epipeda:
            for tmima_id in self._all_tmimata:
                self.print_programma_tmimatos(epipedo_id, tmima_id)

        print('\n# Σχολείο ')
        self.print_programma_sxoleiou()

    print('\nΔιακλάδωση: ', self._solver.NumBranches())
    print('Συγκρούσεις: ', self._solver.NumConflicts())
    print('Χρόνος εκτέλεσης: ', self._solver.WallTime())
```

Προφανώς θα μπορούσε να βρεθεί παραπάνω από ένα πρόγραμμα που ικανοποιεί το πρόβλημα, άρα δημιουργούμε μια κλάση όπου θα επανακαλείται ο επιλυτής και να εκτυπώνει όλες τις δυνατές λύσεις.

```
class
SxolikosProgrammatismosSatSolutionPrinter(cp_model.CpSolverSolutionCallback
):

    def __init__(self):
        cp_model.CpSolverSolutionCallback.__init__(self)
        self.__solution_count = 0

    def OnSolutionCallback(self):
        print(
            f'Λύση #{self.__solution_count}, αντικείμενο:
{self.ObjectiveValue()}'
        )
        self.__solution_count += 1
```

Τέλος για να ξεκινήσει το πρόγραμμα θα πρέπει να αντλήσει όλα τα δεδομένα από ένα .json αρχείο ονομαζόμενο “DATA” που θα είναι αποθηκευμένο στον ίδιο φάκελο όπου είναι και το βασικό αρχείο .py. Αυτό το αναλαμβάνει η συνάρτηση “main”, η οποία αφού αντλήσει και αποθηκεύσει όλα τα απαραίτητα δεδομένα στις κατάλληλες μεταβλητές, θα καλέσει τον επιλυτή και θα αρχίσεις η επίλυση του προβλήματος.

```
def main():


  with open("DATA.json") as DATA:
    #ΦΟΡΤΩΣΗ ΔΕΔΟΜΕΝΩΝ ΑΠΟ ΤΟ JSON
    data = json.load(DATA)

    #ΕΠΙΠΕΔΑ
    EPIPEDA = data["EPIPEDA"]
    print("          ΕΠΙΠΕΔΑ          ")
```

```
print("_____")
for i in range(len(EPIPEDA)):
    print(EPIPEDA[i])

#TMHMATA
TMHMATA = data["TMHMATA"]
print("            TMHMATA            ")
print("_____")
for i in range(len(TMHMATA)):
    print(TMHMATA[i])

#ANTIKEIMENA
dict0 = data["ANTIKEIMENA"]
ANTIKEIMENA = []
for i in range(len(dict0)):
    temp = dict0[i]
    ANTIKEIMENA.append(temp["mathima"])
print("          ANTIKEIMENA          ")
print("-------------------------------")
for i in range(len(ANTIKEIMENA)):
    print(i + 1, ANTIKEIMENA[i])

#ΚΥΚΛΟΣ ΜΑΘΗΜΑΤΩΝ
dict1 = data["KYKLOS_MATHIMATWN"]
KYKLOS_MATHIMATWN = {}
for i in range(len(dict1)):
    temp = dict1[i]
    KYKLOS_MATHIMATWN[temp["epipedo"], temp["antikeimeno"]] = temp["wres"]
print("_____")
print("                ΚΥΚΛΟΣ ΜΑΘΗΜΑΤΩΝ                ")
print("_____")
print("{:<50} {:<40}".format("ΤΑΞΗ, ΑΝΤΙΚΕΙΜΕΝΟ", "ΩΡΕΣ"))
print("_____")
for key, value in KYKLOS_MATHIMATWN.items():
    temp = str(key)
    print("{:<50} {:<40}".format(temp, value))

#ΚΑΘΗΓΗΤΕΣ ΚΑΙ ΜΕΓΙΣΤΕΣ ΩΡΕΣ ΚΑΙ ΕΙΔΙΚΟΤΗΤΕΣ
dict2 = data["KATHIGITES"]
KATHIGITES = {}
temp_dict = {}
EIDIKOTITES = temp_dict.fromkeys(ANTIKEIMENA)

for i in range(len(dict2)):
    temp = dict2[i]
    KATHIGITES[temp["onoma"]] = temp["wres"]

    for j in range(len(dict0)):
        ant = dict0[j]
        if type(temp["eidikotita"]) == str:
            if temp["eidikotita"] == ant["eidikotita"]:
                mathima = ant["mathima"]
                if type(EIDIKOTITES[mathima]) == list:
                    EIDIKOTITES[mathima].append(temp["onoma"])
                else:
                    EIDIKOTITES[mathima] = [temp["onoma"]]
            else:
                pass
        elif type(temp["eidikotita"]) == list:
            for k in range(len(temp["eidikotita"])):
```

```
                if temp["eidikotita"][k] == ant["eidikotita"]:
                    mathima = ant["mathima"]
                    if type(EIDIKOTITES[mathima]) == list:
                        EIDIKOTITES[mathima].append(temp["onoma"])
                    else:
                        EIDIKOTITES[mathima] = [temp["onoma"]]
                else:
                    pass
            else:
                pass

    print("_____")
    print("{:<40} {:<20}".format('ΚΑΘΗΓΗΤΕΣ', 'ΜΕΓΙΣΤΕΣ ΩΡΕΣ'))
    print("_____")
    for key, value in KATHIGITES.items():
        print("{:<40} {:<20}".format(key, value))

    print("_____")
    print("{:<40} {:<20}".format('ΑΝΤΙΚΕΙΜΕΝΟ', 'ΔΙΑΘΕΣΙΜΟΙ ΚΑΘΗΓΗΤΕΣ'))
    print("_____")
    for key, value in EIDIKOTITES.items():
        temp = str(value)
        print("{:<40} {:<20}".format(key, temp))

    #ΧΡΟΝΙΚΕΣ ΠΕΡΙΟΔΟΙ ΚΑΙ ΔΙΑΡΚΕΙΑ
    dict3 = data["Xronikes_Periodoi"]
    XRONIKES_PERIODOI = {}
    for i in range(len(dict3)):
        temp = dict3[i]
        XRONIKES_PERIODOI[temp["Mera-Wres"]] = temp["Diarkeia"]

    print("_____")
    print("{:<40} {:<20}".format('ΜΕΡΑ:    ΩΡΕΣ', 'ΔΙΑΡΚΕΙΑ'))
    print("_____")
    for key, value in XRONIKES_PERIODOI.items():
        print("{:<40} {:<20}".format(key, value))

problem = SxolikosProgrammatismosProblem(EPIPEDA, TMHMATA, ANTIKEIMENA,
KYKLOS_MATHIMATWN, KATHIGITES, EIDIKOTITES,
                                         XRONIKES_PERIODOI)
solver = SxolikosProgrmammatismosSolver(problem)
solver.solve()
```

4.2.3 Οδηγίες εκτέλεσης της εφαρμογής και Παράδειγμα ωρολογίου προγράμματος Γυμνασίου

Σε αυτή την ενότητα θα περιγράψουμε τη διαδικασία που πρέπει να ακολουθήσει κάποιος χρήστης για να μπορέσει να λειτουργήσει τη συγκεκριμένη εφαρμογή. Αρχικά πρέπει να έχει εγκατεστημένο το PyCharm ή κάποιο αντίστοιχο πρόγραμμα καθώς και το OR-Tools όπως είδαμε στην προηγούμενη ενότητα.

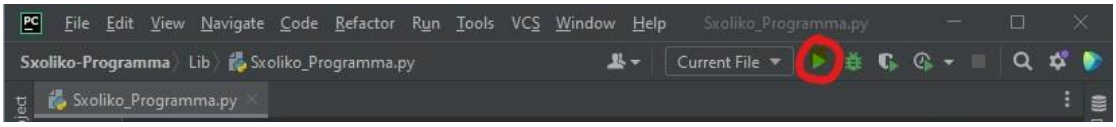
Στη συνέχεια θα χρειαστεί να δημιουργήσει ένα φάκελο στον συγκεκριμένο χώρο «C:\Users\User\PycharmProjects», με το όνομα Sxoliko-Programma και να αντιγράψουμε τα απαραίτητα αρχεία. Το αρχείο Sxoliko_Programma.py είναι το εκτελέσιμο αρχείο, ενώ το αρχείο DATA.json χρειάζεται για να αρχικοποιήσει ο χρήστης τα δεδομένα του.

Το αρχείο .json πρέπει να έχει μια πολύ συγκεκριμένη δομή που παρουσιάζεται με ένα σύντομο παράδειγμα παρακάτω:

```
{
  "EPIPEDA":["A",...],
  "TMHMATA":[1,...],
  "ANTIKEIMENA":[
        {"mathima":"Γλωσσική Διδασκαλία",
        "eidikotita":"Φιλόλογος"
        },...],
"KYKLOS_MATHIMATWN":[
    {"epipedo": "A",
      "antikeimeno": "Γλωσσική Διδασκαλία",
      "wres": 3},...],
"KATHIGITES": [
    {"onoma":"Γιάννης μαθ",
    "wres": 12,
    "eidikotita": "Μαθηματικός"
    },...],
"Xronikes_Periodoi" : [
    {"Mera-Wres": "Δευτέρα : 08:15-09:00",
    "Diarkeia":1
      },...]
}
```

Ο χρήστης πρέπει να συμπληρώσει κατά αυτόν τον τρόπο τα επίπεδα του σχολείου, τα τμήματα, τα αντικείμενα, τον κύκλο μαθημάτων, τους καθηγητές και τις χρονικές περιόδους.

Αφού συμπληρωθεί με τα απαραίτητα δεδομένα το αρχείο DATA.json μετά πρέπει να εκτελεστεί το αρχείο Sxoliko_Programma.py κάνοντας «κλικ» στο τριγωνάκι όπως φαίνεται στην Εικόνα 8.



Εικόνα 8:Εκτέλεση της εφαρμογής

Αμέσως το πρόγραμμα θα αρχίσει την προσπάθεια εύρεσης λύσης. Το αποτέλεσμα θα εμφανιστεί στο πλαίσιο “Python Console”. Παρακάτω ακολουθεί ένα ενδεικτικό παράδειγμα της εκτύπωσης του αποτελέσματος.

ΕΠΙΠΕΔΑ

_____

A …

ΤΜΗΜΑΤΑ

_____

1 …

Κεφάλαιο 4

ΑΝΤΙΚΕΙΜΕΝΑ

_____

1 Γλωσσική Διδασκαλία….

_____

ΚΥΚΛΟΣ ΜΑΘΗΜΑΤΩΝ

_____

ΤΑΞΗ, ΑΝΤΙΚΕΙΜΕΝΟ ΩΡΕΣ

_____

('Α', 'Γλωσσική Διδασκαλία') 3 …..

_____

ΚΑΘΗΓΗΤΕΣ ΜΕΓΙΣΤΕΣ ΩΡΕΣ

_____

Γιάννης 12 …..

_____

ΑΝΤΙΚΕΙΜΕΝΟ ΔΙΑΘΕΣΙΜΟΙ ΚΑΘΗΓΗΤΕΣ

_____

Γλωσσική Διδασκαλία ['Μαρία', 'Γεωργία, 'Μάριος', 'Αθανασία']…..

_____

ΜΕΡΑ: ΩΡΕΣ ΔΙΑΡΚΕΙΑ

_____

Δευτέρα : 08:15-09:00 1

Δευτέρα : 09:05-09:50 1

….

Επίλυση...

Λύση #1, αντικείμενο: 0.0

Κατάσταση: OPTIMAL

 # Καθηγητές

Καθηγητής: Γιάννης

Δευτέρα : 10:55-11:40: Τμήμα:Β-1 Μάθημα:Μαθηματικά

….

Συνολικές ώρες εργασίας: 12h

….

 # Τμήματα

Τμήμα: Α-1

Δευτέρα : 08:15-09:00: Μάθημα: Φυσική Καθηγητής: Αθηνά

….

Συνολικές ώρες εργασίας για Φυσική: 1h

….

 # Σχολείο

Σχολείο:

Δευτέρα : 08:15-09:00:

Α-1:(Φυσική, Αθηνά) Α-2:(Γλωσσική Διδασκαλία, Μαρία) Β-1:(Γλωσσική Διδασκαλία, Γεωργία) Β-2:(Γλωσσική Διδασκαλία, Μάριος) Γ-1:(Μαθηματικά, Γιώργος) Γ-2:(Γλωσσική Διδασκαλία, Αθανασία)

….

Διακλάδωση: 16892

Συγκρούσεις: 0

Χρόνος εκτέλεσης: 3.676958302

Κεφάλαιο 5ο: Συμπεράσματα

Στην παρούσα Πτυχιακή Εργασία αναπτύχθηκε μια εφαρμογή με τη χρήση του λογισμικού Google OR-Tools το οποίο θα δημιουργεί ένα ωρολόγιο σχολικό πρόγραμμα. Όπως παρουσιάστηκε παραπάνω μπορεί με ευκολία να δημιουργήσει ένα ωρολόγιο πρόγραμμα για Δημοτικό, Γυμνάσιο και Λύκειο ή παρόμοιου τύπου επιχειρήσεις όπως είναι τα Φροντιστήρια π.χ. Μέσης Εκπαίδευσης ή Ξένων Γλωσσών. Τα σημαντικότερα προβλήματα που αντιμετωπίζει κανείς στην προσπάθεια να αναπτύξει μια τέτοια εφαρμογή είναι η πληθώρα περιορισμών που υπάρχουν σε καθένα από αυτά τα ιδρύματα. Οπότε η παρούσα εφαρμογή μπορεί να έχει μια κάποια χρησιμότητα μόνο στα πλαίσια της Ελλάδας.

Σίγουρα η παρούσα εφαρμογή δεν παράγει ένα τέλειο ολοκληρωμένο ωρολόγιο πρόγραμμα, όμως σίγουρα θα μπορούσε να βοηθήσει τους εκπαιδευτικούς να έχουν μια πρώτη βάση για να το αναπτύξουν στη συνέχεια. Οι βασικοί περιορισμοί που τέθηκαν στην εργασία ήταν πέντε και ήταν όλοι ισχυροί, δηλαδή έπρεπε να ικανοποιούνται για να πάρουμε κάποιο αποτέλεσμα. Πρακτικά είναι οι απολύτως απαραίτητοι για τη δημιουργία ενός ωρολογίου προγράμματος. Συγκεκριμένα, κάθε Τμήμα πρέπει να έχει τον απαραίτητο αριθμό μαθημάτων που προσδιορίζεται από τον Κύκλο Σπουδών. Κάθε Τμήμα μπορεί να κάνει μόνο ένα μάθημα σε κάθε Περίοδο, που είναι το απολύτως λογικό αφού δεν γίνεται την ίδια χρονική στιγμή ένα άτομο να βρίσκεται σε δύο μέρη ταυτόχρονα. Ο Καθηγητής μπορεί να διδάσκει το πολύ σε ένα Τμήμα τη φορά, με την ίδια λογική όπως και στον προηγούμενο περιορισμό. Επίσης θα πρέπει οι ώρες διδασκαλίας ενός καθηγητή να ξεπερνάν τις μέγιστες ώρες εργασίας κάθε Καθηγητή. Τέλος ο ίδιος Καθηγητής κάνει σε μια Τάξη όλες τις ώρες ενός συγκεκριμένου μαθήματος, δηλαδή να μην διδάσκει στο τμήμα Α1 τα μαθηματικά τη Δευτέρα ο Καθηγητής 1 και την Τρίτη ο Καθηγητής 2.

Προφανώς δεν είναι μόνο αυτοί οι περιορισμοί που χρειάζεται ένα εκπαιδευτικό ίδρυμα. Παρακάτω θα παρατεθούν κάποιοι περιορισμοί που θα μπορούσαν να βελτιώσουν την εφαρμογή. Αρχικά, στην παρούσα εφαρμογή τα τμήματα είναι ίδια για όλα τα επίπεδα, όμως αυτό θα μπορούσε να είναι αλλιώς. Δηλαδή στην Α Γυμνασίου να έχουμε 2 τμήματα και στη Β Γυμνασίου να έχουμε 3 τμήματα. Έπειτα θα μπορούσε να εμπλουτιστεί και με την επιλογή αιθουσών. Για παράδειγμα ανάλογα με τα άτομα του κάθε τμήματος να επιλέγετε μια τάξη, τις ώρες που θα είχαν κάποιο συγκεκριμένο μάθημα να αλλάζει η αίθουσα π.χ. στο μάθημα της Πληροφορικής να ορίζεται η αίθουσα υπολογιστών και η εφαρμογή να προσέχει να μην συμπέσουν τμήματα την ίδια ώρα. Στη συνέχεια υπάρχουν εκπαιδευτικοί που διδάσκουν σε παραπάνω από ένα σχολεία, άρα θα μπορούσε να προστεθεί μια επιλογή για τις μέρες που ένας εκπαιδευτικός επιθυμεί να βρίσκεται στο σχολείο ή ακόμα και ποιες ώρες προτιμάει να βρίσκεται στο σχολείο. Τέλος, θα μπορούσαμε να τονίσουμε ότι κάποιοι από αυτούς τους περιορισμούς θα μπορούσαν να είναι ισχυροί, αλλά κάποιοι άλλοι θα μπορούσαν να είναι «μαλακοί» περιορισμοί, δηλαδή ακόμα και αν δεν ικανοποιούνται θα πάρουμε ένα αποτέλεσμα.

Γενικά υπάρχει μια πληθώρα στην παγκόσμια αλλά και την εγχώρια έρευνα από εργασίας και εφαρμογές που προσπαθούν να επιλύσουν το συγκεκριμένο πρόβλημα, όμως κανένας δεν θα μπορέσει να βρει την τέλεια λύση καθώς ο ανθρώπινος παράγοντας δεν μπορεί να προβλεφθεί πάντα και να καλυφθεί από μια καθολική εφαρμογή. Πάντα θα χρειάζεται η προσαρμογή του ωρολογίου προγράμματος από κάποιον άνθρωπο και ο έλεγχος του, όμως σίγουρα αυταί οι εφαρμογές μπορούν να γλιτώσουν από μεγάλο κόπο και να επιταχύνουν τη διαδικασία σε σημαντικό βαθμό.

Κεφάλαιο 6ο: Βιβλιογραφία

[1] A. Schaerf, «A Survey of Automated Timetabling,» *Artificial Intelligence,* τόμ. 13, pp. 87-127, 1999.

[2] J. Csima, «A class of counterexamples on permanents,» *Pacific Journal of Mathematics,* τόμ. 37, αρ. 3, p. 655–656, 1971.

[3] J. Bondy και U. Murty, Graph Theory with Applications, Ontario, Canada: University of Waterloo, 1976.

[4] D. de Werra, «Equitable colorations of graphs,» *ESAIM: Mathematical Modelling and Numerical Analysis,* τόμ. 5, αρ. 3, pp. 3-8, 1971.

[5] C. Valouxis και E. Housos, «Constraint programming approach for school timetabling,» *Computers & Operations Research,* τόμ. 30, αρ. 10, pp. 1555-1572, 2003.

[6] G. Beligiannis, C. Moschopoulos, G. Kaperonis και S. Likothanassis, «Applying evolutionary computation to the school timetabling problem: The Greek case,» *Computers & Operations Research,* τόμ. 35, αρ. 4, pp. 1265-1280, 2008.

[7] T. Birbas, S. Daskalaki και E. Housos, «School timetabling for quality student and teacher schedules,» *Journal of Scheduling,* τόμ. 12, pp. 177-197, 2009.

[8] I. Tassopoulos και G. Beligiannis, «A hybrid particle swarm optimization based algorithm for high school timetabling problems,» *Applied Soft Computing,* τόμ. 12, p. 3472–3489, 2012.

[9] V. Skoullis, I. Tassopoulos και G. Beligiannis, «Solving the high school timetabling problem using a hybrid cat swarm optimization based algorithm,» *Applied Soft Computing,* τόμ. 52, pp. 277-289, 2017.

[10] E. Demirovic και N. Musliu, «Modeling high school timetabling with bitvectors,» *Annals of Operations Research,* 2017.

[11] J. Kingston, The KHE Timetabling Platform, 2020.

[12] L. Saviniec, M. Santos και A. Costa, «Parallel local search algorithms for high school timetabling problems,» *European Journal of Operational Research,* p. 265, 2017.

[13] G. Fonseca, H. Santos και E. Carrano, «Late acceptance hill-climbing for high school timetabling,» *Journal of Scheduling,* τόμ. 16, p. 453, 2016.

[14] C. Lecoutre, Constraint Networks: Techniques and Algorithms, Wiley, 2013, p. 26.

[15] S. Chandra, C. Gordon, J. Jeannin, C. Schlesinger, M. Sridharan, F. Tip και Y. Choi, Type Inference for Static Compilation of Javascript, Amsterdam, Netherlands: OOPSLA' 16, 2016.

[16] E. Farhi και A. Harrow, Quantum Supremacy through the Quantum Approximate Optimization Algorithm, Cornell Unisversity, 2016.

[17] I. Miguel, Dynamic Flexible Constraint Satisfaction and Its Application to, Department of Computer Science: University of York, 2009.

[18] M. MacDonald και M. Seidenberg, «Constraint Satisfaction Accounts of Lexical and Sentence Comprehension,» σε *Handbook of Psycholinguistics*, 2006.

[19] M. Toro, C. Rueda, C. Agon και G. Assayag, «GELISP: A FRAMEWORK TO REPRESENT MUSICAL,» *Journal of Theoretical and Applied Information Technology,* τόμ. 86, αρ. 2, 2016.

[20] Y. Dong και M. Dong, «Applying constraint satisfaction approach to solve product configuration problems with cardinality-based configuration rules,» *Journal of Intelligent Mnufacturing,* τόμ. 24, pp. 99-111, 2013.

[21] P. Modi, H. Jung, M. Tambe, W. Shen και S. Kulkarni, A Dynamic Distributed Constraint Satisfaction Approach to Resource Allocation, International Conference on Principles and Practice of Constraint Programming: Springer, Berlin, 2001.

[22] S. Russell και P. Norvig, Artificial Intelligence:A Modern Approach, Prentice Hall, 2010.

[23] S. Minton, M. Johnston, A. Philips και P. Laird, «Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems,» *Artificial Intelligence,* τόμ. 58, αρ. 1-3, pp. 161-205, 1992.

[24] R. Sosic και J. Gu, «Efficient local search with conflict minimization: a case study of the n-queens problem,» *IEEE Transactions on Knowledge and Data Engineering,* τόμ. 6, αρ. 5, pp. 661-668, Oct 1994.

[25] R. Sosic και J. Gu, «A Polynomial Time Algorithm for the N-Queens Problem,» *SIGART Bulletin,* τόμ. 1, αρ. 3, pp. 7-11, 1990.

[26] M. Simkin, «The number of n-queens configurations,» *Advances in Mathematics,* p. 427, 2023.

[27] N. Wirth, «The Eight Queens Problem,» *Algorithms and Data Structures,* pp. 114-118, 2004.

[28] G. Gonthier, «Formal Proof — The Four - Color Theorem,» *Notices of the American Mathematical Society,* τόμ. 55, αρ. 11, pp. 1382-1393, 2008.

[29] E. Swart, «The Philosophical Implications of the Four-Color Problem,» *The American Mathematical Monthly,* τόμ. 87, pp. 697-707, 1981.

[30] R. Wilson, «Four colors suffice,» *Princeton Science Library,* 2014.

[31] K. Appel και W. Haken, «Every Planar Map is Four Colorable. I. Discharging,» *Ilinois Journal of Mathematics,* τόμ. 21, αρ. 3, pp. 429-490, 1976.

[32] N. Robertson, D. Sanders, P. Seymour και R. Thomas, «The Four - Colour Theorem,» *Journal of Combinatorial Theory,* τόμ. 70, αρ. 1, pp. 2-44, 1997.

[33] H. Hudson, «Four Colors Do Not Suffice,» *The American Mathematical Monthly,* τόμ. 110, αρ. 5, pp. 417-423, 2003.

[34] K. Appel και W. Haken, «Every Planar Map is Four-Colorable,» *Contemporary Mathematics,* τόμ. 98, 1989.

[35] F. Rossi, P. v. Beek και T. Walsh, Handbook of Constraint Programming, ELSEVIER, 2006.

[36] J. Jaffar και J.-L. Lassez, «Constraint Logic Programming,» *POPL '87: Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages,* pp. 111-119, 1987.

[37] B. Mayoh, E. Tyugu και J. Penjam, Constraint Programming, Springer Science & Business Media, 2013.

[38] G. Lopez, B. Freeman-Benson και A. Borning, «Kaleidoscope: A Constraint Imperative Programming Language,» *NATO ASI CP ,* 1993.

[39] P. Baptiste, C. Le Pape και W. Nuijten, Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems, Springer Science & Business Media, 2012.

[40] C. Bessiere, «Constraint Propagation,» *Handbook of Constraint Programming, Foundations ofArtificial Intelligence,* τόμ. 2, pp. 29-83, 2006.

[41] JetBrains, «JetBrains,» 2023. [Ηλεκτρονικό]. Available: https://www.jetbrains.com/help/pycharm/installation-guide.html.

[42] C. Gonthier, «Formal Proof — The Four Color Theorem,» *Notices of the American Mathematical Society,* pp. 1382-1393, 2008.