UNIVERSITY OF WESTERN MACEDONIA
DEPT. OF INFORMATICS AND TELECOMMUNICATIONS ENGINEERING

Κατασκευή Προγράμματος Εξεταστικής Περιόδου Με Αλγορίθμους Τοπικής Αναζήτησης

Λάμπρου Αθανάσιος

Διπλωματική Εργασία

Επιβλέπων Καθηγητής: Κωνσταντίνος Στεργίου

Κοζάνη 2012

Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, Επίκουρο καθηγητή του τμήματος κύριο Κωνσταντίνο Στεργίου. Η πολύτιμη βοήθεια και καθοδήγησή του σε όλη τη διάρκεια της εργασίας αυτής καθώς και το ενδιαφέρον του για την προσπάθεια αυτή, συνετέλεσαν καταλυτικά στη διαμόρφωση του τελικού αποτελέσματος. Χωρίς τη βοήθειά του η προσπάθεια αυτή δεν θα είχε ολοκληρωθεί.

Στη συνέχεια, νοιώθω την υποχρέωση να ευχαριστήσω την οικογένειά μου και τους φίλους μου για τη στήριξη και τη συμβολή τους σε όλους τους τομείς στην μέχρι τώρα πορεία μου.

Πίνακας Περιεχομένων

Εισαγωγή

Συχνά παρουσιάζονται καταστάσεις οι οποίες θα πρέπει να αντιμετωπιστούν και οι οποίες μπορούν να αναγνωριστούν ως προβλήματα συνδυαστικής βελτιστοποίησης. Η επίλυση ενός προβλήματος συνδυαστικής βελτιστοποίησης συνίσταται στην επιλογή κατάλληλων τιμών για μεγέθη που μπορούν να λάβουν συνεχείς τιμές με στόχο την επίτευξη του καλύτερου δυνατού αποτελέσματος ενώ ταυτόχρονα θα πρέπει να ικανοποιείται ένα σύνολο από περιορισμούς και να βελτιστοποιείται κάποια ποσότητα.. Η απαρίθμηση όλων των πιθανών συνδυασμών τιμών για κάθε μεταβλητή του προβλήματος αποτελεί ένα μη ρεαλιστικό στόχο καθώς παρουσιάζεται το φαινόμενο της συνδυαστικής έκρηξης (combinatorial explosion) σύμφωνα με το οποίο η ύπαρξη πολλών μεταβλητών που ανεξάρτητα η μια από την άλλη μπορούν να λαμβάνουν τιμές δημιουργεί τεράστιο αριθμό συνδυασμών που θα πρέπει να ελεγχθούν. Συνεπώς ο εντοπισμός αποδοτικών μεθόδων που θα είναι σε θέση να επιλύουν ικανοποιητικά, από πλευράς ποιότητας λύσης, απαιτούμενου χρόνου και υπολογιστικών πόρων, προβλήματα συνδυαστικής βελτιστοποίησης αποτέλεσε και συνεχίζει να αποτελεί στόχο εξαιρετικής πρακτικής και ερευνητικής σημασίας.

Διάφοροι κλάδοι της επιστήμης της πληροφορικής αλλά και των μαθηματικών προσέγγισαν με διαφορετικό τρόπο την επίλυση των προβλημάτων συνδυαστικής βελτιστοποίησης. Η επιχειρησιακή έρευνα, η υπολογιστική νοημοσύνη, η τεχνητή νοημοσύνη και οι μεταευρετικές τεχνικές αποτελούν τις γενικές κατηγορίες στις οποίες μπορούν να ομαδοποιηθούν οι περισσότερες τεχνικές επίλυσης προβλημάτων συνδυαστικής βελτιστοποίησης.

Ειδική περίπτωση προβλημάτων συνδυαστικής βελτιστοποίησης αποτελούν τα προβλήματα χρονοπρογραμματισμού. Στη συγκεκριμένη εργασία θα εξεταστεί ένα τέτοιο πρόβλημα χρονοπρογραμματισμού: η δημιουργία του ωρολογίου προγράμματος της εξεταστικής περιόδου. Αρχικά, θα γίνει μια αναφορά στα προβλήματα χρονοπρογραμματισμού γενικότερα, αλλά και στο πρόβλημα του ωρολογίου προγράμματος εξεταστικής περιόδου ειδικότερα (Κεφάλαιο 1). Στη συνέχεια, θα εξεταστούν τα προβλήματα ικανοποίησης περιορισμών, κατηγορία στην οποία εντάσσεται και το συγκεκριμένο πρόβλημα (Κεφάλαιο 2). Κατόπιν, δίδεται μια σύντομη περιγραφή του προβλήματος (Κεφάλαιο 3), ενώ στη συνέχεια αναπτύσσονται οι αλγόριθμοι που χρησιμοποιήθηκαν για την επίλυση αυτού (Κεφάλαιο 4). Σε επόμενο στάδιο ακολουθεί η παρουσίαση της πειραματικής μελέτης που πραγματοποιήθηκε για τη λήψη αποτελεσμάτων, στην οποία θα συμπεριλαμβάνεται περιγραφή και επεξήγηση των αποτελεσμάτων, καθώς και σύγκριση των αποτελεσμάτων αυτών με κάποιο πραγματικό πρόγραμμα εξεταστικής περιόδου (Κεφάλαιο 5). Τέλος, θα παρατεθούν κάποια συμπεράσματα αναφορικά με το πρόβλημα (Κεφάλαιο 6) και μέρος του κώδικα που υλοποιήθηκε.

Κεφάλαιο 1

Προβλήματα timetabling

1.1 Εισαγωγή

Εξετάζοντας τις δραστηριότητες της καθημερινότητας του κάθε ανθρώπου, μπορεί να γίνει αντιληπτό ότι καθημερινό πρόβλημα του καθενός αποτελεί ο χρονοπρογραμματισμός[1], η διαδικασία, δηλαδή, ανάθεσης δραστηριοτήτων σε πόρους στο χρόνο – κατασκευή ενός προγράμματος, ή αλλιώς, «η κατανομή δεδομένων πόρων στη διάρκεια του χρόνου με σκοπό την ολοκλήρωση ενός συνόλου εργασιών». (Baker [1974]) Για παράδειγμα, για τη μεταφορά στην εργασία ή το σχολείο υπάρχουν συγκεκριμένες ώρες που περνάει το λεωφορείο από τη στάση για το συγκεκριμένο προορισμό. Παράλληλα, η διεξαγωγή των διαλέξεων στα πανεπιστήμια ή μαθημάτων στα σχολεία πραγματοποιείται συγκεκριμένες ώρες. Άλλα παραδείγματα αποτελούν οι προκαθορισμένες ώρες απογείωσης και προσγείωσης αεροπλάνων στα αεροδρόμια. Οι προκαθορισμένες ώρες και ημέρες τέλεσης των συνεδριάσεων σε ένα γραφείο. Η παραγωγή προϊόντων σε κάποιο εργοστάσιο έχουν συγκεκριμένη ημερομηνία παράδοσης. Οι βάρδιες των γιατρών και νοσοκόμων στο νοσοκομείο απαιτούν προγραμματισμό έτσι ώστε να καλύπτονται οι ανάγκες των ασθενών.

Όλα τα παραπάνω αποτελούν προβλήματα τα οποία απαιτούν σωστό χρονοπρογραμματισμό, ώστε να μεγιστοποιείται η απόδοση του εκάστοτε οργανισμού. Λύση σε τέτοιου είδους προβλήματα θεωρείται ο προγραμματισμός του συνόλου των διεργασιών σε συγκεκριμένες ώρες, ημέρες και τόπο διεξαγωγής. Η διαδικασία της εύρεσης λύσης εξαρτάται από μια ποικιλία περιορισμών, όπως η διάρκεια των διαδικασιών, η διαθεσιμότητα των πόρων, ο συνολικός διαθέσιμος χρόνος κ.α. Οι παραπάνω περιορισμοί καθορίζουν ένα εύρος αποδεκτών λύσεων για κάθε πρόβλημα και η επιθυμητή λύση ονομάζεται χρονοπρόγραμμα (schedule).

1.2 Προβλήματα Timetabling

Υποκατηγορία των προβλημάτων προγραμματισμού αποτελούν τα "προβλήματα ανάθεσης" (assignment problems)[4]. Ο στόχος των προβλημάτων ανάθεσης είναι να βρεθεί μια βέλτιστη ανάθεση πόρων σε δραστηριότητες χωρίς να χρησιμοποιηθεί ένας πόρος περισσότερες από μία φορά και να εξασφαλιστεί ότι όλες οι δραστηριότητες ολοκληρώνονται. Ο στόχος μπορεί να είναι να ελαχιστοποιηθεί ο συνολικός χρόνος, ή να ολοκληρωθεί ένα σύνολο στόχων, ή να ελαχιστοποιηθεί το κόστος των αναθέσεων. Τα προβλήματα timetabling είναι μια ειδική περίπτωση των προβλημάτων ανάθεσης. Ο σκοπός των προβλημάτων timetabling είναι "η κατανομή δοθέντων πόρων σε αντικείμενα, που τοποθετούνται στον χώρο/χρόνο με τέτοιο τρόπο ώστε να ικανοποιούνται όσο το δυνατό περισσότεροι επιθυμητοί στόχοι, και όλα αυτά να γίνονται με την προϋπόθεση ότι ικανοποιούνται κάποιοι αυστηροί

περιορισμοί". Τα προβλήματα αυτά λόγω της πρακτικής και καθημερινής τους σημασίας βρίσκουν μεγάλη εφαρμογή στην καθημερινή ζωή.

1.3 Αναπαράσταση και Δυσκολίες Προβλημάτων Timetabling

Γενικά, ένα πρόβλημα timetabling μπορεί να αναπαρασταθεί σαν ένα σύνολο Ε γεγονότων, ένα σύνολο S σχισμών και ένα σύνολο C περιορισμών ανάμεσα στα γεγονότα. Στόχος είναι ο καθορισμός των γεγονότων σε χρονοθυρίδες με τέτοιο τρόπο ώστε να ικανοποιούνται όλοι οι περιορισμοί. Με τον όρο χρονοθυρίδα εννοούμε μια αυθαίρετη δομή δεδομένων που αναπαριστά τον χρόνο όπως έχει οριστεί αυτός στην μοντελοποίηση του προβλήματος. Συγκεκριμένα, τα προβλήματα παραγωγής ωρολογίου προγράμματος εξεταστικής περιόδου (Examination Timetabling Problems- ETTP) που θα εξεταστούν στη παρούσα εργασία, αναπαριστώνται από το σύνολο Ε των εξεταστέων μαθημάτων, το σύνολο S χρονοθυρίδων ή περιόδων και το σύνολο C περιορισμών που σχετίζονται με την ανάθεση εξεταστέων μαθημάτων σε περιόδους. .Για την παρούσα μοντελοποίηση μια χρονοθυρίδα αποτελείται από τον συνδυασμό ημέρας, ώρας, αίθουσας εξέτασης, όνομα μαθήματος, όνομα καθηγητή και εξάμηνο κατά το οποίο διδάσκεται το μάθημα. Για παράδειγμα, η ακολουθία (1,2,4,S1,T1,1) δείχνει ότι την πρώτη ημέρα(Δευτέρα) ,στο δεύτερο τρίωρο (12:00-15:00) της ημέρας, και στη τέταρτη αίθουσα, θα εξεταστεί το μάθημα S1 που το διδάσκει ο T1 .Επίσης το S1 είναι μάθημα του πρώτου εξαμήνου. Τέτοιου είδους προβλήματα ανήκουν στην κλάση των προβλημάτων NP-πλήρη(NP-complete). Αυτό όπως φανερώνεται στη συνέχεια χαρακτηρίζει τα προβλήματα παραγωγής ωρολογίου προγράμματος ως ιδιαίτερα δύσκολα.

Άλλη μια δυσκολία των συγκεκριμένων προβλημάτων είναι το γεγονός ότι βασίζονται σε πολλές παραμέτρους. Για τη δημιουργία του ωρολογίου προγράμματος εξεταστικής περιόδου λαμβάνουμε υπόψη το σύνολο των μαθημάτων προς εξέταση, το σύνολο των διαθέσιμων ημερών (εβδομάδων), το σύνολο των διαθέσιμων ωρών εξέτασης ανά μέρα καθώς και το σύνολο των διαθέσιμων αιθουσών για να πραγματοποιηθούν οι εξετάσεις. Έτσι, για παράδειγμα, αν υποθέσουμε ότι έχουμε 40 μαθήματα προς εξέταση κατά τη διάρκεια 15 ημερών (3 εβδομάδων) με 4 τρίωρα και 4 αίθουσες διαθέσιμες κάθε μέρα, βλέπουμε ότι η πολυπλοκότητα του προβλήματος (εκφρασμένη ως το πλήθος των πιθανών συνδυασμών) είναι $(15*4*4)^{30}=240^{30}$. Όπως βλέπουμε, η πολυπλοκότητα είναι αρκετά μεγάλη, γεγονός που δείχνει τη δυσκολία του προβλήματος αυτού.

Ένας τρίτος λόγος για τη δυσκολία των προβλημάτων αυτών είναι ότι πολλές φορές τα προβλήματα αυτά περιπλέκονται κατά άγνωστο βαθμό, ο οποίος όμως εξαρτάται από τα χαρακτηριστικά και τις λεπτομέρειες του προβλήματος. Έτσι, κάποιος αλγόριθμος που μπορεί να βρίσκει λύση για κάποιο συγκεκριμένο πρόβλημα, ενδέχεται να μην μπορεί να το κάνει για κάποιο παρόμοιο πρόβλημα που εμπεριέχει κάποιους ειδικούς περιορισμούς. Ένας τελευταίος λόγος για τη δυσκολία των προβλημάτων αυτών είναι πως πολλά προβλήματα του πραγματικού κόσμου περιέχουν περιορισμούς οι οποίοι δεν μπορούν να αναπαρασταθούν αποδοτικά με ακρίβεια.

1.4 Κατηγορίες Προβλημάτων Timetabling

Έχουν προταθεί πολλές κατηγοριοποιήσεις για τα προβλήματα timetabling. Σύμφωνα με τον A. Schaeft [6] τα προβλήματα timetabling μπορούν να κατηγοριοποιηθούν ανάλογα με τον τύπο του οργανισμού στον οποίο αναφέρονται και τον τύπο των περιορισμών τους, στις τρεις παρακάτω κατηγορίες:

Σχολικό timetabling: αφορά τον εβδομαδιαίο προγραμματισμό για όλες τις τάξεις ενός σχολείου, αποφεύγοντας κάποιος καθηγητής να διδάσκει ταυτόχρονα δύο μαθήματα και αντίθετα.

Ακαδημαϊκό timetabling: αφορά τον εβδομαδιαίο προγραμματισμό όλων των διαλέξεων σε ένα σετ μαθημάτων πανεπιστημίου, ελαχιστοποιώντας τις αλληλεπικαλύψεις διαλέξεων μαθημάτων που έχουν κοινούς φοιτητές.

Timetabling εξεταστικής: αφορά τον προγραμματισμό για τις εξετάσεις ενός σετ μαθημάτων πανεπιστημίου, αποφεύγοντας αλληλεπικαλύψεις εξετάσεων μαθημάτων που έχουν κοινούς φοιτητές και απλώνοντας τις εξετάσεις όσο το δυνατό περισσότερο.

Όπως παρατηρήθηκε από τον Schaeft[6], η κατηγοριοποίηση αυτή δεν είναι ακριβής, αφού κάποια συγκεκριμένα προβλήματα μπορούν να ενταχθούν κάπου ενδιάμεσα από τις παραπάνω κατηγορίες, αδυνατώντας να τοποθετηθούν σε κάποια από αυτές επακριβώς. Ο Carter[7], στη μελέτη του για τις πρόσφατες εξελίξεις στο πρακτικό timetabling εξετάσεων αναγνώρισε πέντε διαφορετικά υποπροβλήματα για το πρόβλημα timetabling μαθημάτων:

- Timetabling μαθημάτων
- Timetabling αίθουσας-καθηγητή
- Προγραμματισμός μαθητών
- Ανάθεση καθηγητών
- Ανάθεση αίθουσας

Όπως παρατηρούμε έχουν δοθεί πολλές κατηγοριοποιήσεις για τα συγκεκριμένα προβλήματα και ο λόγος είναι ότι τα προβλήματα timetabling διαφέρουν πολύ μεταξύ τους. Οι λόγοι για τους οποίους διαφέρουν είναι οι παρακάτω:

- Τα προβλήματα timetabling διαιρούνται ανάλογα με τον βαθμό ελευθερίας στην τοποθέτηση παροχής πόρων και τα διαστήματα ζήτησης των πόρων. Στα *καθαρά προβλήματα* η τοποθέτηση των πόρων δεν πρέπει να περνάει τη διαθέσιμη χωρητικότητα. Στα *προβλήματα διανομής πόρων* είναι γνωστό εκ των προτέρων πόσοι και τι είδους πόροι θα χρειαστούν και η τοποθέτηση πρέπει να είναι ίδια, ή και να ξεπερνά, τη διαθέσιμη χωρητικότητα.
- Διαφορετικά περιβάλλοντα έχουν διαφορετικούς περιορισμούς, οι οποίοι συνεισφέρουν στη πολυπλοκότητα του προβλήματος.
- Το μέγεθος του προβλήματος μπορεί να ποικίλει από μερικές δραστηριότητες (μικρά πεδία τιμών για τις μεταβλητές) μέχρι χιλιάδες δραστηριότητες. Έτσι, λόγω πολυπλοκότητας, οι αλγόριθμοι που εφαρμόζονται επιτυχώς στα μικρά προβλήματα δεν μπορούν να εφαρμοστούν στα πιο πολύπλοκα.
- Σημαντικά αριθμητικά χαρακτηριστικά των προβλημάτων διαφέρουν σε διαφορετικά περιβάλλοντα. Επίσης, στο ίδιο περιβάλλον διαφέρουν από την ξεχωριστή περίπτωση αλγορίθμου επίλυσης.
- Ανάλογα με το περιβάλλον ο χρόνος απόκρισης για τη δημιουργία χρονοπρογράμματος μπορεί να διαφέρει από μερικά δευτερόλεπτα σε κάποιες μέρες

- Τέλος, πολλές φορές απαιτείται η τροποποίηση του χρονοπρογράμματος όταν το περιβάλλον είναι μεταβλητό ή αλλάζει λόγω ανθρώπινης παρέμβασης.

1.5 Περιγραφή Προβλήματος Δημιουργίας Ωρολογίου Προγράμματος Εξεταστικής Περιόδου.

Όπως αναφέρθηκε και παραπάνω, στόχος του προγράμματος της εξεταστικής περιόδου είναι να ανατεθούν όλα τα διαθέσιμα μαθήματα προς εξέταση σε κατάλληλες μέρες, ώρες και αίθουσες εξετάσεις, ώστε να μην παραβιάζεται κανένας από τους περιορισμούς που έχουν τεθεί. Η τοποθέτησή τους, δηλαδή, σε κατάλληλες χρονικές στιγμές ή αλλιώς χρονοθυρίδες.

Οι περιορισμοί που θέτονται κατά τη δημιουργία κάθε προγράμματος εξεταστικής περιόδου, μπορεί να διαφέρουν από πανεπιστήμιο σε πανεπιστήμιο, ή και από τμήμα σε τμήμα. Οι λόγοι είναι διάφοροι. Για παράδειγμα, ο συνολικός αριθμός μαθημάτων διαφέρει από σχολή σε σχολή. Ακόμα, σε κάποια μαθήματα απαιτείται διαφορετικός τρόπος εξέτασης, όπως για παράδειγμα ορισμένα μαθήματα που απαιτούν εξέταση και σε εργαστήριο.

Οι περιορισμοί που τίθενται σε κάθε πρόγραμμα χρονοπρογραμματισμού, άρα και στο δικό μας, χωρίζονται σε δύο κατηγορίες:

Δεσμευτικοί (Hard): οι δεσμευτικοί περιορισμοί είναι εκείνοι οι οποίοι πρέπει να τηρούνται αναγκαστικά, για να οδηγηθούμε σε επιτρεπτό αποτέλεσμα. Τέτοιοι περιορισμοί είναι για παράδειγμα η μη εξέταση δύο μαθημάτων την ίδια μέρα, την ίδια ώρα και στην ίδια αίθουσα. Έτσι θα αποφευχθούν συγκρούσεις οι οποίες οδηγούν σε μη πραγματοποιήσιμο πρόγραμμα στο φυσικό κόσμο. Τα προγράμματα τα οποία δεν παραβιάζουν κανένα δεσμευτικό περιορισμό ονομάζονται *αληθοφανή*.

Επιθυμητοί (Soft): οι επιθυμητοί περιορισμοί καθορίζουν, σε ένα βαθμό, την ποιότητα του τελικού αποτελέσματος (προγράμματος). Είναι περιορισμοί, η παραβίαση των οποίων δεν εμποδίζει τη δημιουργία του τελικού αποτελέσματος, αλλά η τήρηση των οποίων καταλήγει σε πιο αποδοτικό και ποιοτικό αποτέλεσμα. Παράδειγμα τέτοιου περιορισμού είναι η πιθανή επιθυμία κάποιου καθηγητή, να εξεταστούν τα μαθήματα του μόνο Δευτέρα. Όσο περισσότεροι επιθυμητοί περιορισμοί ικανοποιούνται, τόσο πιο ποιοτικό είναι το αποτέλεσμα.

Ένα καλής ποιότητας πρόγραμμά θεωρείται εκείνο το οποίο είναι αληθοφανές και ικανοποιεί όσο το δυνατό περισσότερους επιθυμητούς περιορισμούς. Βέβαια, επειδή η έννοια της ποιότητας είναι υποκειμενική, η παραπάνω συνθήκη δεν καταλήγει πάντα σε ποιοτικά αποδεκτό αποτέλεσμα για όλους. Η εύρεση ενός αποτελέσματος το οποίο θα είναι αποδεκτό από όλους είναι δύσκολη. Ένα κοινό μέτρο που συναντάται συχνά σε πολλές εφαρμογές ορίζεται σύμφωνα με τον αριθμό των συγκρουόμενων περιορισμών. Ο τρόπος με τον οποίο οι τελευταίοι χειρίζονται μπορεί να ποικίλλει από πανεπιστήμιο σε πανεπιστήμιο ενώ είναι δύσκολο να επιτευχθεί ένας γενικός μαθηματικός τύπος για αυτά τα προβλήματα. Στη παρούσα εργασία θα ασχοληθούμε μόνο με τους δεσμευτικούς περιορισμούς.

1.6 Μαθηματικό μοντέλο.

Το μαθηματικό μοντέλο, στο οποίο θα στηριχτεί η δημιουργία του ωρολογίου προγράμματος εξεταστικής περιόδου είναι το παρακάτω. Δεχόμαστε ότι έχουμε m

συνολικό αριθμό μαθημάτων προς εξέταση και x συνολικό αριθμό διαθέσιμων χρονοθυρίδων (x = συνολικές μέρες εξέτασης * ώρες εξέτασης κάθε μέρας * αίθουσες εξέτασης). Το C αποτελεί το κόστος μιας συνολικής ανάθεσης τιμών σε όλες τις μεταβλητές (ο αριθμός των περιορισμών που παραβιάζονται για ανάθεση χρονοθυρίδας σε κάθε μάθημα i). Επίσης θα ισχύει ότι $Y_{ik}=1$, αν έχει ανατεθεί η τιμή k στη μεταβλητή i, διαφορετικά $Y_{ik}=0$. Ακόμα, θα ισχύει ότι $X_{ij}=1$, αν υπάρχει σύγκρουση λόγω παραβίασης κάποιου περιορισμού ανάμεσα στα μαθήματα i και j, διαφορετικά $X_{ij}=0$. Έτσι, η δημιουργία του προγράμματος θα βασιστεί στις παρακάτω τρεις εξισώσεις:

$C = 0$ (1)

$\sum_{k=1}^{x} Y_{ik} = 1$ για κάθε μάθημα i (2)

$\sum_{i=1}^{m} \sum_{j=1}^{m} X_{ij} = 0$ (3)

Η πρώτη εξίσωση μας δείχνει το κόστος όλων των αναθέσεων των μαθημάτων σε χρονοθυρίδες. Στόχος του προβλήματος είναι η μείωση αυτής της τιμής όσο το δυνατό περισσότερο. Η δεύτερη εξίσωση μας εξασφαλίζει ότι όλα τα μαθήματα έχουν ανατεθεί σε ακριβώς μία χρονοθυρίδα το καθένα. Τέλος, η Τρίτη εξίσωση διασφαλίζει ότι δεν παραβιάζεται κανένας περιορισμός.

1.7 Τρόποι επίλυσης προβλήματος.

Η πλειοψηφία των παλαιότερων τεχνικών (βλέπε Schmidt και Strohlein, 1979) ήταν βασισμένες σε προσομοίωση του ανθρώπινου τρόπου επίλυσης του προβλήματος. Όλες αυτές οι τεχνικές, τις οποίες ονομάζουμε *απευθείας ευρεστικές*, ήταν βασισμένες στη διαδοχική αύξηση. Αυτό σημαίνει ότι ένα μερικό ωρολόγιο πρόγραμμα εξεταστικής επεκτείνεται σταδιακά, μάθημα με μάθημα, έως ότου όλα τα μαθήματα έχουν προγραμματιστεί. Η βασική ιδέα όλων αυτών των μεθόδων είναι να προγραμματίζεται πάντα πρώτα το πιο περιορισμένο μάθημα. Η διαφορά τους έγκειται στο πως ορίζει ο καθένας την έκφραση «το πιο περιορισμένο μάθημα».

Αργότερα, ερευνητές άρχισαν να εφαρμόζουν γενικές τεχνικές στο συγκεκριμένο πρόβλημα. Σαν αποτέλεσμα, βλέπουμε αλγορίθμους βασισμένους στο προγραμματισμό ακέραιων αριθμών, στο network flow κ.α. Επιπρόσθετα, το πρόβλημα αντιμετωπίστηκε απλοποιώντας το σε ένα γνωστό πρόβλημα, το χρωματισμό γράφου.

Πιο πρόσφατα, κάποιες προσεγγίσεις βασισμένες σε τεχνικές αναζήτησης που χρησιμοποιούνται στην τεχνητή νοημοσύνη εμφανίστηκαν στη βιβλιογραφία. Μεταξύ άλλων έχουμε τους αλγορίθμους ικανοποίησης περιορισμών. Άλλη μια κατηγορία αλγορίθμων είναι οι μετα-ευριστικοί (meta-heuristic), χαρακτηριστικό των οποίων είναι ότι ξεκινούν από ένα αρχικό πρόβλημα και διεξάγουν τεχνικές έρευνας οι οποίες προσπαθούν να αποφύγουν τοπικά ελάχιστα ή μέγιστα. Παραδείγματα τέτοιων αλγορίθμων είναι οι simulated annealing, tabu search και οι γενετικοί αλγόριθμοι.

Στη συγκεκριμένη εργασία θα ερευνήσουμε τεχνικές εύρεσης λύσης, δίνοντας έμφαση στις πιο πρόσφατες προσεγγίσεις γενικά και στις προσεγγίσεις της τεχνητής νοημοσύνης ειδικότερα.

Κεφάλαιο 2

Προβλήματα ικανοποίησης περιορισμών

2.1 Εισαγωγή

Τα τελευταία χρόνια, ο *προγραμματισμός με περιορισμούς* (Constraint Programming, CP) έχει κερδίσει την προσοχή των ειδικών εξαιτίας της δυνατότητάς του να λύνει πολύ δύσκολα προβλήματα του πραγματικού κόσμου. Όχι μόνο είναι βασισμένος σε ένα ισχυρό θεωρητικό υπόβαθρο, αλλά προσελκύει ευρύ εμπορικό ενδιαφέρον επίσης. Ειδικότερα, στις περιοχές προβλημάτων μοντελοποίησης ετερογενούς βελτιστοποίησης και ικανοποίησης. Πρόσφατα, αναγνωρίστηκε από την ACM (Association for Computing Machinery) ως μία από τις στρατηγικές κατευθύνσεις τις υπολογιστικής έρευνας. Ωστόσο, την ίδια στιγμή, ο προγραμματισμός με περιορισμούς είναι μια από τις λιγότερο γνωστές και κατανοητές τεχνολογίες.

Οι περιορισμοί προκύπτουν στις περισσότερες περιοχές της ανθρώπινης καθημερινότητας. Τυποποιούν τις εξαρτήσεις στο φυσικό κόσμο. Περιορισμός είναι απλά μια λογική σχέση ανάμεσα σε μεταβλητές, κάθε μια από τις οποίες παίρνει μια τιμή σε ένα δοσμένο πεδίο τιμών. Ο περιορισμός, κατά συνέπεια, περιορίζει τις πιθανές τιμές ενδιαφέροντος. Οι περιορισμοί μπορούν επίσης να είναι ετερογενείς, ώστε να περιορίζουν τιμές από διαφορετικά πεδία, για παράδειγμα, το μήκος (αριθμός) με τη λέξη (συμβολοσειρά). Το σημαντικό χαρακτηριστικό των περιορισμών είναι ο δηλωτικός τους χαρακτήρας, με την έννοια ότι καθορίζουν τι συσχέτιση πρέπει να ισχύει, χωρίς να διευκρινίζουν μια υπολογιστική διαδικασία για να εφαρμοστεί αυτή η συσχέτιση. Συγκεντρωτικά, οι ιδιότητες των περιορισμών μπορούν να συνοψισθούν παρακάτω:

- Οι περιορισμοί δίνουν μερική πληροφορία. Δεν προσδιορίζουν μοναδιαία την τιμή κάποιας μεταβλητής. Για παράδειγμα, ο περιορισμός X>5 δεν υπολογίζει ποια ακριβώς τιμή θα πάρει η μεταβλητή X.
- Οι περιορισμοί είναι ετερογενείς, με την έννοια ότι μπορούν να συσχετίσουν μεταβλητές με διαφορετικά πεδία τιμών.
- Οι περιορισμοί είναι μη-κατευθυνόμενοι. Δεδομένου ενός περιορισμού ανάμεσα στις μεταβλητές X και Y, αυτός μπορεί να χρησιμοποιηθεί για να την εύρεση της τιμής X μέσω της τιμής του Y, αλλά και το αντίθετο.
- Οι περιορισμοί είναι δηλωτικοί. Απεικονίζουν το είδος της σχέσης ανάμεσα σε δύο μεταβλητές αλλά δεν δίνουν καμία πληροφορία για το πώς θα επιλυθεί αυτή η σχέση.
- Οι περιορισμοί είναι προσθετικοί. Η σειρά με την οποία δηλώνονται ή λαμβάνονται υπόψη από το πρόβλημα δεν επηρεάζει τη λύση του.
- Οι περιορισμοί ενός προβλήματος σπάνια είναι ανεξάρτητοι μεταξύ τους, καθώς οι μεταβλητές συνήθως συμμετέχουν σε παραπάνω από ένα περιορισμούς.

Όλοι μας χρησιμοποιούμε περιορισμούς ως ένα σημείο κλειδί της καθημερινότητάς μας. “Θα είμαι εκεί από τις πέντε ως τις έξι” είναι ένα τυπικό

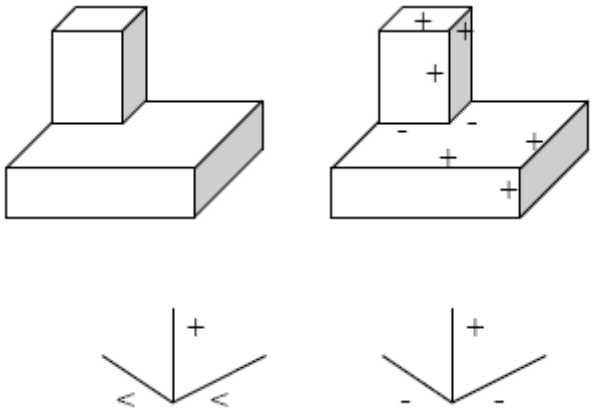
παράδειγμα περιορισμού που χρησιμοποιούμε για να καθορίσουμε το χρόνο μας. Φυσικά, δεν λύνουμε μόνο ένα περιορισμό, αλλά μια συλλογή περιορισμών που σπάνια είναι ανεξάρτητοι, πράγμα που περιπλέκει λίγο το πρόβλημα.

Ο *προγραμματισμός περιορισμών* είναι η μελέτη υπολογιστικών συστημάτων βασισμένων σε περιορισμούς. Η ιδέα του προγραμματισμού περιορισμών είναι να λύνει προβλήματα θέτοντας περιορισμούς (απαιτήσεις) για την περιοχή του προβλήματος και κατά συνέπεια να βρίσκει λύση που ικανοποιεί αυτούς τους περιορισμούς.

"Ο προγραμματισμός περιορισμών αντιπροσωπεύει μία από τις κοντινότερες προσεγγίσεις που έχει κάνει ως τώρα η υπολογιστική επιστήμη προς τη κατεύθυνση του Ιερού Δισκοπότηρου του προγραμματισμού: ο χρήστης δίνει ένα πρόβλημα και ο υπολογιστής το λύνει." [E. Freuder]

2.2 Ιστορικά στοιχεία

Οι πρώιμες ιδέες, που κατέληξαν στον προγραμματισμό περιορισμών μπορούν να βρεθούν στη τεχνητή νοημοσύνη στις δεκαετίες του 60 και του 70. Το πρόβλημα *μαρκαρίσματος σκηνής* (scene labelling) είναι πιθανώς το πρώτο πρόβλημα ικανοποίησης περιορισμών που τυποποιήθηκε. Ο στόχος είναι να αναγνωριστούν τρισδιάστατα αντικείμενα με την ερμηνεία των γραμμών σε δυσδιάστατα σχέδια. Αρχικά, οι γραμμές (ή ακμές) του δυσδιάστατου σχήματος πρέπει να κατηγοριοποιηθούν σε κυρτές(+), κοίλες(-), γραμμές που περικλείουν το σχήμα προς τα δεξιά(>) και γραμμές που περικλείουν το σχήμα προς τα αριστερά(<). Πρέπει να πούμε πως η κατηγοριοποίηση μιας γραμμής ως κυρτή ή κοίλη εξαρτάται από την σκοπιά που βλέπει ο παρατηρητής το αντικείμενο. Σε ποιο ανεπτυγμένα συστήματα, υπάρχει η δυνατότητα αναγνώρισης κομματιών του σχεδίου με σκιά. Στο παρακάτω σχήμα μπορούμε να δούμε ένα παράδειγμα της συγκεκριμένης τεχνικής:



Οι κύριοι αλγόριθμοι που αναπτύχθηκαν σε εκείνη τη περίοδο (όπως ο αλγόριθμος μαρκαρίσματος του Waltz[8]) αφορούσαν την επίτευξη κάποιας μορφής συνέπειας.

Μια άλλη εφαρμογή για τους περιορισμούς είναι η *διαλογική γραφική παράσταση* (interactive graphics) όπου το Sketchpad του Ivan Sutherland [9], αναπτυγμένο στις αρχές της δεκαετίας του '60, ήταν το πρωτοποριακό σύστημα. Το Sketchpad και το ακόλουθό του, ThingLab[10] από τον Alan Borning, ήταν διαλογικές εφαρμογές γραφικής παράστασης που επέτρεψαν στο χρήστη να σχεδιάζει και να επεξεργάζεται

περιορισμένες γεωμετρικές φιγούρες στην οθόνη του υπολογιστή. Αυτά τα συστήματα συμβάλλουν στην ανάπτυξη των τοπικών μεθόδων διάδοσης και της σύνταξης περιορισμών.

Το κύριο βήμα προς το προγραμματισμό περιορισμών επιτεύχθηκε όταν οι Gallaire, Jaffar και Lassez σημείωσαν ότι ο λογικός προγραμματισμός ήταν ένα είδος προγραμματισμού περιορισμών. Η βασική ιδέα πίσω από το λογικό προγραμματισμό (Logical Programming, LP), και το δηλωτικό προγραμματισμό (Declarative programming) γενικότερα, είναι ότι ο χρήστης δηλώνει *τι* πρέπει να λυθεί αντί το *πώς* να λυθεί, πράγμα το οποίο είναι πολύ κοντά στην ιδέα των περιορισμών. Επομένως ο συνδυασμός περιορισμών και λογικού προγραμματισμού είναι πολύ φυσικός και ο λογικός προγραμματισμός περιορισμών (Constraint Logical Programming, CLP) δημιουργεί ένα φυσικό δηλωτικό περιβάλλον για την επίλυση προβλημάτων με τη βοήθεια των περιορισμών. Εντούτοις, δεν σημαίνει ότι ο προγραμματισμός περιορισμών είναι περιορισμένος στο CLP. Οι περιορισμοί ενσωματώθηκαν σε χαρακτηριστικές γλώσσες όπως C++ και την Java επίσης.

Οι σημερινές πραγματικές εφαρμογές του CP στον τομέα του προγραμματισμού, του σχεδιασμού και της βελτιστοποίησης θέτουν την ερώτηση εάν ο παραδοσιακός τομέας της έρευνας επιχειρησιακών διαδικασιών (Operations Research, OR) είναι ανταγωνιστής ή συνεταίρος του CP. Ενώ η OR έχει μια μακροχρόνια ερευνητική παράδοση και μια πολύ επιτυχή μέθοδο για να λύνει προβλήματα χρησιμοποιώντας γραμμικό προγραμματισμό, ο CP δίνει έμφαση στις μοντελοποιήσεις και μεθόδους εύρεσης λύσης υψηλότερου επιπέδου, οι οποίες είναι ευκολότερες στη κατανόηση από τον τελικό πελάτη.

Οι πρόσφατη πρόοδος στο τομέα υπόσχεται ότι και οι δύο μεθοδολογίες μπορούν να εκμεταλλευτούν η μια την άλλη, και ιδιαίτερα ο προγραμματισμός περιορισμών μπορούν να χρησιμοποιηθεί ως μια πλατφόρμα για την ενσωμάτωση διάφορων αλγορίθμων επίλυσης περιορισμών, συμπεριλαμβανομένων και εκείνων που αναπτύχθηκαν και αποδείχτηκε ότι είναι επιτυχείς μέσα από την επιχειρησιακή έρευνα.

Από τις παραπάνω παραγράφους, μπορούμε να δούμε ότι ο προγραμματισμός περιορισμών συνδυάζει και εκμεταλλεύεται ιδέες από ένα μεγάλο εύρος πεδίων, όπως η τεχνητή νοημοσύνη, οι συνδυαστικοί αλγόριθμοι, η υπολογιστική λογική, τα διακριτά μαθηματικά, τα νευρωνικά δίκτυα, η επιχειρησιακή έρευνα και οι γλώσσες προγραμματισμού.

2.3 Ορισμός

Τα προβλήματα ικανοποίησης περιορισμών είναι αντικείμενο της τεχνητής νοημοσύνης εδώ και χρόνια. Ένα *πρόβλημα ικανοποίησης περιορισμών* (Constraint Satisfaction Problem, CSP) ορίζεται ως:

- Ένα σετ μεταβλητών $X=\{x_1, \ldots, x_n\}$
- Για κάθε μεταβλητή x_i, ένα σετ πεπερασμένο σετ D_i πιθανών τιμών (το πεδίο ορισμού του) και
- Ένα σετ *περιορισμών* που περιορίζει τις τιμές που μπορούν να πάρουν ταυτόχρονα οι μεταβλητές.

Να σημειώσουμε ότι οι τιμές των πεδίων δεν χρειάζεται να είναι ένα σετ συνεχόμενων ακέραιων (αν και τις περισσότερες φορές είναι). Δεν χρειάζεται να είναι καν αριθμητικές τιμές.

Ως λύση ενός CSP ορίζεται η ανάθεση τιμών σε όλες τις μεταβλητές, από το πεδίο ορισμού τους, με τέτοιο τρόπο, ώστε να ικανοποιούνται όλοι οι περιορισμοί ταυτόχρονα. Ανάλογα με το πρόβλημα, μπορεί να χρειαζόμαστε:

- Μόνο μία λύση, χωρίς ιδιαίτερη προτίμηση για το ποια,
- Όλες τις λύσεις,
- Μια βέλτιστη, ή τουλάχιστον πολύ καλή λύση, δοσμένης μιας αντικειμενικής συνάρτησης που ορίστηκε με βάση κάποιες ή όλες τις μεταβλητές.

Λύσεις σε ένα CSP μπορούν να βρεθούν ψάχνοντας (συστηματικά) μέσω των πιθανών αναθέσεων τιμών σε μεταβλητές. Οι μέθοδοι έρευνας χωρίζονται σε δύο κύριες κατηγορίες: αυτές που διαχειρίζονται μερικές λύσεις (ή μερικές αναθέσεις τιμών) και αυτές που εξερευνούν πλήρεις αναθέσεις τιμών σε όλες τις μεταβλητές στοχαστικά.

2.4 Ικανοποίηση περιορισμών.

2.4.1 Συστηματική αναζήτηση.

Από θεωρητικής σκοπιάς, η λύση προβλημάτων ικανοποίησης περιορισμών με χρήση συστηματικής εξερεύνησης του χώρου των λύσεων είναι τετριμμένη. Αντίθετα, πρακτικά, αν και οι μέθοδοι συστηματικής έρευνας (χωρίς επιπλέον βελτιώσεις) φαίνονται πολύ απλές και μη αποδοτικές είναι πολύ σημαντικές γιατί αποτελούν τη βάση για πιο εξελιγμένους και αποδοτικούς αλγορίθμους.

Ο βασικό αλγόριθμος για CSPs, ο οποίος ψάχνει στο χώρο των πλήρων αναθέσεων, είναι ο *generate-and-test* (GT). Η ιδέα του GT είναι απλή. Αρχικά, γίνεται μια πλήρης ανάθεση τιμών σε όλες τις μεταβλητές (τυχαία) και στη συνέχεια, αν η ανάθεση αυτή ικανοποιεί όλους τους περιορισμούς, τότε η λύση έχει βρεθεί. Διαφορετικά γίνεται μια νέα ανάθεση τιμών στις μεταβλητές.

Ο αλγόριθμος GT είναι ένας αδύναμος αλγόριθμος που χρησιμοποιείται εάν αποτύχουν όλοι οι υπόλοιποι. Η αποδοτικότητά του είναι φτωχή εξαιτίας μη ενημερωμένης γεννήτριας και αργής διαπίστωσης ασυνεπειών. Επομένως, δύο τρόποι αύξησης της αποδοτικότητας του είναι οι:

Η γεννήτρια αποτιμήσεων είναι «έξυπνη». Με τον όρο αυτό εννοούμε πως παράγει μια πλήρης αποτίμηση με τέτοιο τρόπο ώστε οι συγκρούσεις που βρίσκονται στη φάση ελέγχου να ελαχιστοποιούνται. Αυτή είναι η ιδέα των στοχαστικών αλγορίθμων που βασίζονται στη τοπική αναζήτηση.

Η γεννήτρια είναι συγχωνευμένη με τον δοκιμαστή, δηλαδή, η εγκυρότητα του κάθε περιορισμού ελέγχεται αμέσως μόλις λάβουν τιμές οι μεταβλητές που εμπλέκονται σε αυτόν. Αυτή η μέθοδος χρησιμοποιείται στην προσέγγιση με υπαναχώρηση (backtracking).

Η υπαναχώρηση (BT) είναι η μέθοδος λύσης CSP που επεκτείνει αυξητικά μια μερική λύση (ορισμένες μεταβλητές έχουν λάβει τιμές συνεπείς προς τους περιορισμούς) του προβλήματος προς την ολική. Ο τρόπος λειτουργίας της είναι, επαναλαμβανόμενα, να επιλέγει μια τιμή για μια νέα μεταβλητή η οποία θα είναι συνεπής με τις τιμές της τρέχουσας μερικής λύσης.

Όπως αναφέρθηκε και παραπάνω, μπορούμε να δούμε την BT σαν μια συγχώνευση των φάσεων παραγωγής και ελέγχου του αλγορίθμου GT. Αναθέτονται τιμές στις μεταβλητές σειριακά και μόλις όλες οι μεταβλητές κάποιου περιορισμού

έχουν πάρει τιμή, γίνεται έλεγχος για την εγκυρότητά του. Εάν η μερική λύση παραβιάζει κάποιον από τους περιορισμούς, πραγματοποιείται υπαναχώρηση στη πιο πρόσφατα αρχικοποιημένη μεταβλητή που έχει ακόμα εναλλακτικές αρχικοποιήσεις. Γίνεται αντιληπτό ότι, όποτε μια μερική αρχικοποίηση παραβιάζει κάποιον περιορισμό, η υπαναχώρηση μπορεί να διαγράψει ένα υποχώρο του καρτεσιανού προϊόντος όλων των πεδίων ορισμού των μεταβλητών. Κατά συνέπεια, η υπαναχώρηση είναι πολύ καλύτερη από την μέθοδο GT, αν και η πολυπλοκότητά της για τα περισσότερα προβλήματα είναι και πάλι εκθετικού βαθμού.

Υπάρχουν τρία μειονεκτήματα της καθιερωμένης (χρονολογικής) υπαναχώρησης:

- Το thrashing: η συνεχόμενη αποτυχία λόγο του ίδιου λάθους.
- Η περιττή εργασία, αφού οι συγκρουόμενες μεταβλητές τιμών δεν συγκρατούνται στη μνήμη
- Η αργή εύρεση της σύγκρουσης, αφού η σύγκρουση δεν μπορεί να γίνει αντιληπτή πριν συμβεί.

Έχουν προταθεί μέθοδοι για την αντιμετώπιση των δύο πρώτων μειονεκτημάτων όπως οι μέθοδοι *Backjumping* και *Backmarking*, αλλά περισσότερη προσοχή δόθηκε στο να ανιχνεύονται οι ασυνέπειες νωρίτερα με χρήση *τεχνικών συνέπειας*.

2.4.2 Τεχνικές συνέπειας.

Μια άλλη προσέγγιση για την λύση CSP είναι βασισμένη στην αφαίρεση ασυνεπών τιμών από τα πεδία ορισμού των μεταβλητών μέχρι την εύρεση της λύσης. Αυτές οι μέθοδοι ονομάζονται *τεχνικές συνέπειας* και παρουσιάστηκαν για πρώτη φορά στο πρόβλημα μαρκαρίσματος σκηνής (scene labeling)[12]. Να σημειώσουμε ότι οι τεχνικές συνέπειας είναι μη-ντετερμινιστικές σε αντίθεση με την ντετερμινιστική αναζήτηση.

Υπάρχουν αρκετές μέθοδοι συνέπειας[13] αλλά οι περισσότερες από αυτές δεν είναι πλήρεις. Επομένως, οι τεχνικές συνέπειας σπάνια χρησιμοποιούνται μόνες τους για τη λύση ενός CSP.

Τα ονόματα των βασικών τεχνικών συνέπειας προέρχονται από έννοιες γράφων. Το CSP αντιπροσωπεύεται συνήθως ως γράφος (δίκτυο) περιορισμών, όπου οι κόμβοι αντιστοιχούν στις μεταβλητές και οι ακμές αντιστοιχούν στους περιορισμούς. Η συγκεκριμένη διαδικασία απαιτεί το CSP να έχει μια συγκεκριμένη μορφή, γνωστή ως *δυαδικό CSP* (binary CSP), δηλαδή να περιέχει μοναδιαίους και δυαδικούς περιορισμούς μόνο. Έχει δειχτεί ότι αυθαίρετα CSPs μπορούν να μετατραπούν σε δυαδικά, αλλά η διαδικασία της δυαδικοποίησης πιθανότατα δεν θα αξίζει τον κόπο αφού οι αλγόριθμοι μπορούν εύκολα να επεκταθούν για να μπορούν να χειριστούν μη δυαδικά CSPs.

Η πιο απλή τεχνική συνέπειας είναι η *συνέπεια κόμβου* (node consistency, NC). Απλά αφαιρεί τιμές από τα πεδία ορισμού μεταβλητών οι οποίες είναι ασυνεπείς με μοναδιαίους περιορισμούς των αντίστοιχων μεταβλητών.

Η πιο διαδεδομένη τεχνική συνέπειας είναι η *συνέπεια τόξου* (arc consistency, AC). Αυτή η τεχνική αφαιρεί τιμές από τα πεδία τιμών μεταβλητών οι οποίες δεν είναι συνεπείς με δυαδικούς περιορισμούς. Συγκεκριμένα, το τόξο (V_i, V_j) είναι συνεπές αν και μόνο αν για κάθε τιμή x του τρέχοντος πεδίου ορισμού του Vi που ικανοποιεί τους περιορισμούς του V_i, υπάρχει κάποια τιμή y στο πεδίο ορισμού του V_j ώστε αν V_i=x και V_j=y, να μην παραβιάζεται ο δυαδικός περιορισμός ανάμεσα στο V_i και στο V_j.

Υπάρχουν αρκετοί αλγόριθμοι συνέπειας τόξου, που ξεκινούν από τον AC-1 και καταλήγουν κάπου στον AC-7. Οι αλγόριθμοι αυτοί είναι βασισμένοι στις επαναληπτικές αναθεωρήσεις τόξων έως ότου να επιτευχθεί μια συνεπής κατάσταση να γίνει το πεδίο τιμής κάποιας μεταβλητής άδειο. Οι πιο δημοφιλείς ανάμεσά του είναι οι AC-3 και AC-4. Ο AC-3 πραγματοποιεί αναθεωρήσεις μόνο στα τόξα τα οποία είναι πιθανό να επηρεάστηκαν από μια προηγούμενη αναθεώρηση. Δεν απαιτεί κάποια ειδική δομή δεδομένων, σε αντίθεση με τον AC-4, ο οποίος λειτουργεί με μεμονωμένα ζευγάρια τιμών για να αφαιρέσει πιθανή ανεπάρκεια που προκύπτει από τον επαναλαμβανόμενο έλεγχο ζευγαριών. Χρειάζεται ειδική δομή δεδομένων για να λειτουργήσει, για να μπορεί να «θυμάται» τα ζευγάρια των ασυνεπών τιμών των ασυνεπών μεταβλητών και είναι, συνεπώς, λιγότερο αποδοτικός σε θέματα μνήμης από τον AC-3.

Ακόμα περισσότερες ασυνεπείς τιμές μπορούν να αφαιρεθούν με τη χρήση τεχνικών *συνέπειας μονοπατιού* (Path Consistency, PC). Η συνέπεια μονοπατιού απαιτεί, για κάθε ζευγάρι τιμών δύο μεταβλητών X και Y, οι οποίες ικανοποιούν τον αντίστοιχο δυαδικό περιορισμό, να υπάρχει τιμή για κάθε μεταβλητή σε κάποιο μονοπάτι ανάμεσα στη X και στη Y, τέτοια ώστε όλοι οι δυαδικοί περιορισμοί σε αυτό το μονοπάτι να ικανοποιούνται. Έχει δειχτεί από τον Montanary[14] ότι ένα CSP έχει συνέπεια μονοπατιού αν και μόνο αν κάθε μονοπάτι του με μήκος δύο έχει συνέπεια μονοπατιού. Επομένως, οι αλγόριθμοι συνέπειας μονοπατιού μπορούν να εργαστούν με τρειπλέτες μεταβλητών (μονοπάτια μεγέθους δύο). Υπάρχουν αρκετοί τέτοιοι αλγόριθμοι όπως ο PC-1 και ο PC-2, αλλά χρειάζονται εκτεταμένη απεικόνιση των περιορισμών, πράγμα που κοστίζει πολύ σε μνήμη.

Όλες οι παραπάνω τεχνικές συνέπειας καλύπτονται από μια γενική έννοια της K-συνέπειας και της ισχυρής k-συνέπειας. Ένας γράφος περιορισμών είναι k-συνεπής αν, για κάθε συνδυασμό τιμών για K-1 μεταβλητές που ικανοποιούν όλους τους περιορισμούς των μεταβλητών αυτών, υπάρχει μια τιμή για την αυθαίρετη K-οστή μεταβλητή ώστε όλοι οι περιορισμοί ανάμεσα στις K μεταβλητές να ικανοποιούνται. Ένας γράφος περιορισμών είναι ισχυρά K-συνεπής αν είναι J-συνεπής για κάθε $J \leq K$.

Συμπληρώνοντας τα προηγούμενα, μπορούμε να πούμε ότι:

- Η συνέπεια κόμβου είναι ισοδύναμη με ισχυρή 1-συνέπεια.
- Η συνέπεια τόξου είναι ισοδύναμη με ισχυρή 2-συνέπεια.
- Η συνέπεια μονοπατιού είναι ισοδύναμη με ισχυρή 3-συνέπεια.

Έχουν δημιουργηθεί αλγόριθμοι για να κάνουν ένα γράφο περιορισμών ισχυρά K-συνεπή για $K>2$, αλλά πρακτικά χρησιμοποιούνται σπάνια λόγω θεμάτων απόδοσης. Αν και αυτοί οι αλγόριθμοι αφαιρούν περισσότερες ασυνέπειες από κάθε αλγόριθμο συνέπειας τόξου, δεν εξαλείφουν την ανάγκη για αναζήτηση. Μπορούμε να δούμε, ότι αν ένας γράφος περιορισμών με N κόμβους είναι ισχυρά N-συνεπής, τότε η λύση στο CSP μπορεί να βρεθεί χωρίς αναζήτηση. Αλλά, η χειρότερη δυνατή πολυπλοκότητα σε ένα τέτοιο γράφο είναι εκθετική. Δυστυχώς, εάν ένας γράφος περιορισμών είναι ισχυρά K-συνεπής για κάποιο $K<N$, τότε, γενικά, η αναζήτηση με υπαναχώρηση δεν μπορεί να αποφευχθεί, ακόμα υπάρχουν, δηλαδή, ασυνεπείς τιμές.

Επειδή οι περισσότερες τεχνικές συνέπειας (NC, AC, PC) δεν είναι πλήρεις, με την έννοια ότι αφήνουν κάποιες ασυνεπείς τιμές, οι περιορισμένες μορφές αυτών των τεχνικών αφαιρούν ίδιες ποσότητες ασυνεπειών, αλλά είναι πιο αποδοτικές. Για παράδειγμα, η *κατευθυνόμενη συνέπεια τόξου* (Directional Arc Consistency, DAC) επισκέπτεται κάθε τόξο μόνο μια φορά και έτσι απαιτεί λιγότερο υπολογισμό από την AC-3 και λιγότερη μνήμη από την AC-4. Παρόλα αυτά, η DAC μπορεί να επιτύχει πλήρη συνέπεια τόξου για πολλά προβλήματα.

2.4.3 Διάδοση περιορισμών.

Και οι συστηματικές τεχνικές αναζήτησης και (μερικές) τεχνικές συνέπειας μπορούν να χρησιμοποιηθούν μόνες τους για να λύσουν CSP εντελώς, αλλά αυτό γίνεται σπάνια. Ο συνδυασμός των δύο προσεγγίσεων είναι ένας πιο κοινός τρόπος λύσης CSP.

Οι τεχνικές *Look Back* χρησιμοποιούν ελέγχους συνέπειας σε ήδη αρχικοποιημένες μεταβλητές. Η αναζήτηση με υπαναχώρηση (BT) είναι ένα απλό παράδειγμα αυτής της κατηγορίας τεχνικών. Για να αποφευχθούν κάποια προβλήματα της BT, όπως το thrashing και η περιττές εργασίες, έχουν προταθεί άλλες παρόμοιες τεχνικές.

Η *υπαναχώρηση με άλμα* (Backjumping, BJ) είναι μια μέθοδος που χρησιμοποιούμε για να αποφύγουμε το thrashing της BT, τη συνεχόμενη αποτυχία λόγω του ίδιου σφάλματος. Η λειτουργία της BJ είναι ίδια με της BT, εκτός από το σημείο που πραγματοποιείται η υπαναχώρηση. Και οι δύο αλγόριθμοι επιλέγουν μια μεταβλητή κάθε φορά και ψάχνουν για μια τιμή για την μεταβλητή αυτή, σιγουρεύοντας ότι η νέα ανάθεση είναι συμβατή με τις αναθέσεις που έχουν γίνει μέχρι εκείνο το σημείο. Ωστόσο, αν η BJ βρει μια ασυνέπεια, αναλύει την κατάσταση για να εντοπίσει την πηγή της ασυνέπειας αυτής. Χρησιμοποιεί τους περιορισμούς που παραβιάζονται για να εντοπίσει τη μεταβλητή που ευθύνεται για τις συγκρούσεις. Αν όλες οι τιμές του πεδίου ορισμού έχουν ελεγχθεί, τότε επιστρέφει στη πιο πρόσφατα συγκρουόμενη μεταβλητή. Αυτή είναι η βασική διαφορά με τη BT, οποία επιστρέφει πάντα στη αμέσως προηγούμενη μεταβλητή.

Άλλοι αλγόριθμοι της κατηγορίας Look Back είναι οι *Backchecking* (BC) και *Backmarking* (BM)[15]. Στόχος τους είναι να εξαλείψουν τη περιττή εργασία που γίνεται στη BT. Τόσο ο Backchecking, όσο και ο απόγονός του Backmarking είναι χρήσιμοι αλγόριθμοι για τη μείωση των ελέγχων συμβατότητας. Για παράδειγμα, αν ο αλγόριθμος εντοπίσει ότι η ανάθεση Y/b (ανάθεση της τιμής b στη μεταβλητή Y) δεν είναι συμβατή με καμία πρόσφατη ανάθεση X/a (ανάθεση της τιμής a στη μεταβλητή X), τότε κρατάει στη μνήμη αυτή την ασυμβατότητα. Έτσι, εφόσον η X/a είναι δεσμευμένη, η Y/b δεν θα εξεταστεί πάλι.

Ο Backmarking είναι μια βελτίωση του Backchecking που αποφεύγει τόσο περιττούς ελέγχους περιορισμών, όσο και περιττές ανακαλύψεις ασυνεπειών. Μειώνει τον αριθμό ελέγχων συμβατότητας, κρατώντας στη μνήμη, για κάθε τρέχουσα ανάθεση, τις μη συμβατές πρόσφατες αναθέσεις. Επιπλέον, αποφεύγει τους επαναλαμβανόμενους ελέγχους συμβατότητας, οι οποίοι έχουν ήδη γίνει κ ήταν επιτυχείς.

Όλοι οι αλγόριθμοι της κατηγορίας Look Back έχουν κοινό μειονέκτημα τον αργό εντοπισμό της σύγκρουσης. Λύνουν την ασυνέπεια αφού έχει συμβεί αλλά δεν τη προλαβαίνουν πριν συμβεί. Επομένως, προτάθηκαν αλγόριθμοι που εντάχθηκαν στη κατηγορία *Look Ahead*, οι οποίοι προλαβαίνουν τις ασυνέπειες πριν δημιουργηθούν καν.

Ο *πρώιμος έλεγχος* (Forward Checking, FC) είναι το πιο απλό παράδειγμα αλγορίθμων αυτής της κατηγορίας. Πραγματοποιεί ελέγχους συνέπειας τόξου σε ζευγάρια μεταβλητών που αποτελούνται από μια μεταβλητή δεν έχει αρχικοποιηθεί ακόμα και μία που έχει. Πρακτικά, όταν ανατίθεται μια τιμή στη τρέχουσα μεταβλητή, οποιαδήποτε τιμή στο πεδίου ορισμού κάποιας μελλοντικής μεταβλητής η οποία συγκρούεται με αυτή την ανάθεση, αφαιρείται (προσωρινά) από το πεδίο ορισμού. Επομένως, ο FC διατηρεί τη σταθερότητα ότι για κάθε μεταβλητή που δεν έχει αρχικοποιηθεί ακόμα, υπάρχει τουλάχιστον μία τιμή στο πεδίο ορισμού της, η

οποία να είναι συμβατή με τις αναθέσεις τιμών στις μεταβλητές που έχουν αρχικοποιηθεί μέχρι εκείνη τη στιγμή. Ο FC κάνει περισσότερη δουλειά από την BT όταν κάθε ανάθεση προστίθεται στη τρέχουσα μερική λύση και, γενικά, είναι πάντα καλύτερη επιλογή από τη χρονολογική υπαναχώρηση.

Ακόμα περισσότερες μελλοντικές ασυνέπειες αφαιρούνται με τη μέθοδο *μερικής προεξέτασης* (Partial Look Ahead, PLA). Σε αντίθεση με τον FC, που πραγματοποιεί ελέγχους στους περιορισμούς της τρέχουσας μεταβλητής και των μελλοντικών μεταβλητών, ο PLA επεκτείνει αυτό τον έλεγχο συνέπειας στις μεταβλητές που δεν έχουν απευθείας σύνδεση με τις αρχικοποιημένες μεταβλητές, χρησιμοποιώντας κατευθυνόμενη συνέπεια τόξου.

Η προσέγγιση που χρησιμοποιεί πλήρης συνέπεια τόξου μετά από κάθε βήμα ανάθεσης ονομάζεται *πλήρης προεξέταση* ((Full) Look Ahead, LA) ή αλλιώς *διατήρηση συνέπειας τόξου* (Maintaining Arc Consistency, MAC). Μπορεί να χρησιμοποιήσει αλγόριθμο συνέπειας τόξου για να επιτύχει συνέπεια, ωστόσο, πρέπει να σημειωθεί ότι ο LA κάνει περισσότερη δουλειά από τον FC και η PLC, όταν κάθε ανάθεση προστίθεται στη τρέχουσα μερική λύση. Στη πραγματικότητα, ο LA, σε μερικές περιπτώσεις, μπορεί να είναι πιο "ακριβός" υπολογιστικά από την BT και γι' αυτό ο FC και η BT χρησιμοποιούνται ακόμα σε εφαρμογές.

2.4.3 Στοχαστικοί και ευριστικοί αλγόριθμοι.

Μέχρι στιγμής, παρουσιάσαμε αλγορίθμους ικανοποίησης περιορισμών που επεκτείνουν μια μερική συνεπής λύση σε μια πλήρης ανάθεση, η οποία ικανοποιεί όλους τους περιορισμούς. Τα τελευταία χρόνια, άπληστες στρατηγικές τοπικής αναζήτησης έχουν γίνει δημοφιλείς ξανά. Αυτοί οι αλγόριθμοι μεταβάλλουν αυξητικά τις ασυνεπείς τιμές σε όλες τις μεταβλητές. Χρησιμοποιούν τη μέθοδο "hill climbing" για να κινηθούν προς όσο πιο πλήρεις λύσεις μπορούν. Για να αποφύγουν να κολλήσουν σε διάφορα τοπικά βέλτιστα, είναι εφοδιασμένοι με διάφορες ευρεστικές τεχνικές που τυχαιοποιούν την αναζήτηση. Η στοχαστική τους φύση γενικά αποβάλει τη "πληρότητα" που προσφέρουν οι συστηματικοί μέθοδοι αναζήτησης.

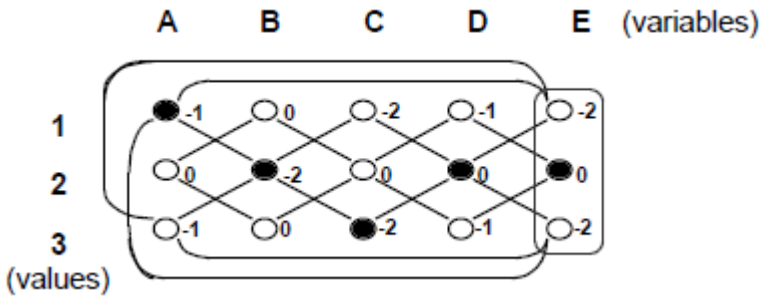
Ο "hill climbing" είναι πιθανότατα ο πιο γνωστός αλγόριθμος τοπικής αναζήτησης[16]. Ξεκινά από μια τυχαία ανάθεση τιμών και σε κάθε βήμα του αλλάζει τη τιμή κάποιας μεταβλητής, με τέτοιο τρόπο ώστε η νέα ανάθεση να ικανοποιεί περισσότερους περιορισμούς. Σε περίπτωση που ο αλγόριθμος κολλήσει σε κάποιο τοπικό βέλτιστο, επανεκκινεί από μια νέα τυχαία ανάθεση τιμών. Ο αλγόριθμος τερματίζει μόλις βρει ολικό βέλτιστο, δηλαδή όταν όλοι οι περιορισμοί ικανοποιούνται. Να σημειωθεί ότι ο συγκεκριμένος αλγόριθμος ερευνά πολλές γειτονικές καταστάσεις της τρέχουσας πριν προβεί σε κάποια κίνηση.

Για να αποφευχθεί η αναζήτηση όλης της όλων των γειτονικών καταστάσεων, προτάθηκε το ευρετικό min-conflicts (MC). Η μέθοδος αυτή επιλέγει τυχαία κάποια από της μεταβλητές που συγκρούονται, κάποια μεταβλητή δηλαδή που εμπλέκεται σε περιορισμό που παραβιάζεται και επιλέγει μια τιμή που μειώνει τον αριθμό των περιορισμών που παραβιάζονται. Αν δεν υπάρχει τέτοια τιμή, επιλέγει τυχαία μια τιμή που δεν αυξάνει τον αριθμό των περιορισμών που παραβιάζονται (η τρέχουσα τιμή επιλέγεται μόνο αν όλες οι υπόλοιπες αυξάνουν των αριθμό των συγκρούσεων).

Επειδή ο απλός min-conflicts δεν μπορεί να υπερβεί κάποιο τοπικό ελάχιστο, ορισμένες στρατηγικές προστέθηκαν στον MC. Ανάμεσα σε αυτές, η random-walk (RW) έχει γίνει μία από τις πιο δημοφιλείς. Για μια δοθείσα συγκρουόμενη μεταβλητή, η random-walk επιλέγει τυχαία μια τιμή με πιθανότητα p και εφαρμόζει

το MC με πιθανότητα 1-p. Το random-walk μπορεί να εφαρμοστεί και στον αλγόριθμο hill-climbing και το αποτέλεσμα είναι ο Steepest-Descend-Random-Walk (SDRW) αλγόριθμος.

Μια ακόμα μέθοδος που κάνει έρευνα στο χώρο των πλήρων αναθέσεων τιμών έως ότου να βρεθεί λύση είναι βασισμένη στη *διασύνδετη προσέγγιση* που απεικονίζεται από τον GENET[18] αλγόριθμο. Το πρόβλημα ικανοποίησης περιορισμών αναπαριστάται από ένα δίκτυο όπου οι κόμβοι αντιστοιχούν στις τιμές των μεταβλητών. Οι κόμβοι που αντιστοιχούν στις τιμές μια μεταβλητής ομαδοποιούνται σε ένα cluster και υποτίθεται ότι ακριβώς ένας κόμβος ενεργοποιείται στο συγκεκριμένο cluster, πράγμα το οποίο σημαίνει ότι η τιμή αυτή έχει επιλεγεί για τη μεταβλητή. Υπάρχει μια ανασταλτική σύνδεση (τόξο) ανάμεσα σε δύο κόμβους διαφορετικών clusters που αντιστοιχούν σε ζεύγος συγκρουόμενων τιμών σύμφωνα με τους περιορισμούς που υπάρχουν ανάμεσα σ' αυτές τις μεταβλητές.



Ο αλγόριθμος ξεκινά με μια τυχαία διαμόρφωση του δικτύου και επαναϋπολογίζει την κατάσταση των κόμβων σε ένα cluster επαναλαμβανόμενα, λαμβάνοντας υπόψη μόνο την κατάσταση των γειτονικών κόμβων και τα βάρη των συνδέσεων με αυτούς. Όταν ο αλγόριθμος φτάσει σε μια σταθερή διαμόρφωση του δικτύου, η οποία δεν είναι λύση του προβλήματος, μπορεί να οδηγηθεί σε λύση χρησιμοποιώντας ένα απλό κανόνα εκμάθησης ο οποίος ενδυναμώνει τα βάρη των συνδέσεων που αντιστοιχούν σε παραβιασμένους περιορισμούς. Όπως όλοι οι παραπάνω στοχαστικοί αλγόριθμοι, έτσι και ο GENET είναι ατελής.

2.5 Βελτιστοποίηση περιορισμών.

Σε πολλές πραγματικές εφαρμογές, δεν θέλουμε να βρεθεί μια οποιαδήποτε λύση αλλά μια καλή λύση. Η ποιότητα της λύσης μετριέται συνήθως από μια, εξαρτώμενη από την εφαρμογή, συνάρτηση αποκαλούμενη αντικειμενική συνάρτηση. Ο στόχος είναι να βρεθεί τέτοια λύση που να ικανοποιεί όλους τους περιορισμούς και να ελαχιστοποιεί ή να μεγιστοποιεί την αντικειμενική συνάρτηση αντίστοιχα. Τέτοια προβλήματα αναφέρονται ως προβλήματα βελτιστοποίησης ικανοποίησης περιορισμού (Constrain Satisfaction Optimization Problems - CSOP).

Ένα CSOP αποτελείται από ένα τυποποιημένο CSP και μια συνάρτηση βελτιστοποίησης που χαρτογραφεί κάθε λύση (πλήρες ανάθεση τιμών στις μεταβλητές) σε μια αριθμητική αξία[19].

Ο πιο ευρέως χρησιμοποιημένος αλγόριθμος για να βρίσκει βέλτιστες λύσεις ονομάζεται Branch and Bound (B&B)[20] και μπορεί να εφαρμοστεί σε CSOP επίσης. Ο B&B χρειάζεται μια ευρετική συνάρτηση, η οποία να αντιστοιχεί μια μερική ανάθεση τιμών σε κάποια αριθμητική αξία. Αντιπροσωπεύει μια κατώτερη εκτίμηση

(σε περίπτωση ελαχιστοποίησης) της αντικειμενικής συνάρτησης για τη καλύτερη πλήρης ανάθεση τιμών που λαμβάνεται από τη συγκεκριμένη μερική ανάθεση. Ο αλγόριθμος ψάχνει για λύσεις κατά βάθος και συμπεριφέρεται όπως το χρονολογικό BT εκτός από το ότι όταν μια τιμή ορίζεται στη μεταβλητή, υπολογίζεται η τιμή της ευρετικής συνάρτησης για την ανάθεση αυτή. Εάν αυτή η αξία υπερβαίνει κάποιο όριο, το υπο-δέντρο κάτω από τη τρέχουσα μερική ανάθεση κλαδεύεται αμέσως. Αρχικά, το όριο τίθεται στο (συν) άπειρο και κατά τη διάρκεια του υπολογισμού καταγράφεται η τιμή της καλύτερης λύσης που έχει βρεθεί μέχρι εκείνη τη στιγμή.

Η αποδοτικότητα του B&B καθορίζεται από δύο παράγοντες: τη ποιότητα της ευρετικής συνάρτησης και εάν ένα καλό όριο μπορεί να βρεθεί γρήγορα. Παρατηρήσεις από πραγματικά προβλήματα έδειξαν επίσης ότι το «τελευταίο βήμα» προς τη βέλτιστη λύση, δηλαδή η βελτίωση μιας καλής λύσης ακόμη περισσότερο, είναι συνήθως το πιο υπολογιστικά ακριβό μέρος της διαδικασίας λύσης. Ευτυχώς, σε πολλές εφαρμογές, οι χρήστες ικανοποιούνται με μια λύση που είναι κοντά στο βέλτιστο εάν αυτή η λύση βρίσκεται σύντομα. Ο B&B αλγόριθμος μπορεί επίσης να χρησιμοποιηθεί για να βρεθούν μη βέλτιστες λύσεις με τη χρήση το δεύτερο όριο «αποδοχής». Εάν ο αλγόριθμος βρει μια λύση που είναι καλύτερη από το επιτρεπτό όριο, αυτή η λύση μπορεί να επιστραφεί στο χρήστη ακόμα κι αν δεν είναι βέλτιστη.

2.6 Εφαρμογές.

Ο προγραμματισμός περιορισμών έχει εφαρμοστεί επιτυχώς σε πολλές διαφορετικές προβληματικές περιοχές τόσο διαφορετικές μεταξύ τους όσο την ανάλυση της δομής του DNA, τον χρονοπρογραμματισμό των νοσοκομείων ή το σχεδιασμό των βιομηχανιών. Έχει αποδείξει ότι έχει προσαρμοστεί καλά στην επίλυση πραγματικών προβλημάτων.

Τα προβλήματα ανάθεσης ήταν ίσως ο πρώτος τύπος βιομηχανικής εφαρμογής που επιλύθηκε με εργαλεία περιορισμών. Ένα χαρακτηριστικό παράδειγμα είναι η κατανομή θέσεων στάθμευσης για τους αερολιμένες, όπου τα αεροσκάφη πρέπει να σταθμεύσουν στη διαθέσιμη στάση κατά τη διάρκεια της παραμονής στον αερολιμένα (αερολιμένας Roissy στο Παρίσι) ή η κατανομή γκισέ για τις αίθουσες αναχώρησης (διεθνής αερολιμένας Χονγκ Κονγκ). Ένα άλλο παράδειγμα είναι η κατανομή αγκυροβολίων για σκάφη στο λιμάνι (διεθνή τερματικά Χονγκ Κονγκ.

Ένας άλλος χαρακτηριστικός τομέας εφαρμογής περιορισμών είναι η ανάθεση προσωπικού όπου οι κανόνες και οι κανονισμοί εργασίας επιβάλλουν δύσκολους περιορισμούς. Η σημαντική πτυχή σε αυτά τα προβλήματα είναι η απαίτηση να ισορροπηθεί η εργασία μεταξύ διάφορων προσώπων. Συστήματα όπως το Gymnaste[22] αναπτύχθηκαν για την παραγωγή των καταλόγων για τις νοσοκόμες στα νοσοκομεία, για την ανάθεση πληρωμάτων στις πτήσεις (SAS, British Airways, Swissair) ή την ανάθεση προσωπικού στις επιχειρήσεις σιδηροδρόμων (S.N.C.F, Italian Railway Company)[23].

Πιθανώς ο επιτυχέστερος τομέας εφαρμογής για περιορισμούς με πεπερασμένο πεδίο είναι τα προβλήματα προγραμματισμού, όπου, πάλι, οι περιορισμοί εκφράζουν φυσικά τα πραγματικά όρια. Λογισμικό βασισμένο στους περιορισμούς χρησιμοποιείται για τον προγραμματισμό καλής λειτουργίας, το σχεδιασμό δασικής επεξεργασίας, το σχεδιασμό παραγωγής στη βιομηχανία πλαστικού (InSol) ή για τον προγραμματισμό παραγωγής μεταφορικών και στρατιωτικών αεροπλάνων (Dassault Aviation). Η χρήση περιορισμών σε συστήματα προχωρημένου σχεδιασμού και

προγραμματισμού αυξάνεται λόγω της τρέχουσας τάσης για παραγωγή κατόπιν παραγγελίας.

Μια άλλη μεγάλη περιοχή της εφαρμογής περιορισμών είναι η διαχείριση και διαμόρφωση δικτύων. Αυτά τα προβλήματα περιλαμβάνουν τον προγραμματισμό της τοποθέτησης καλωδίων των δικτύων τηλεπικοινωνιών σε κάποιο κτήριο (France Telecom) ή τον επανασχεδιασμό δικτύων ηλεκτροδότησης για τον σχεδιασμό συντήρησης χωρίς να διακόπτονται οι υπηρεσίες των πελατών (Ether). Ένα άλλο παράδειγμα είναι η βέλτιστη τοποθέτηση σταθμών βάσης στα ασύρματα εσωτερικά δίκτυα τηλεπικοινωνιών.

Υπάρχουν πολλές άλλες περιοχές από έχουν αντιμετωπιστεί χρησιμοποιώντας τους περιορισμούς. Πρόσφατες εφαρμογές περιλαμβάνουν τα γραφικά υπολογιστών (έκφραση της γεωμετρικής συνοχής στην περίπτωση της ανάλυσης εικόνας, σχεδιασμός προγραμμάτων, διεπαφές χρήστη), την επεξεργασία φυσικής γλώσσας (κατασκευή των αποδοτικών κατατμητών), συστήματα βάσεων δεδομένων (για να εξασφαλιστεί ή/και να αποκατασταθεί η συνέπεια των στοιχείων), τη μοριακή βιολογία (αλληλουχία DNA), τις επιχειρηματικές εφαρμογές (εμπορικές συναλλαγές), την ηλεκτρική μηχανική (εντοπισμός ελαττωμάτων), τον σχεδιασμό κυκλωμάτων (για τον υπολογισμό των διατάξεων), για προβλήματα μεταφοράς κτλ.

2.7 Περιορισμοί.

Η εκτενής χρήση εφαρμογής του προγραμματισμού περιορισμών στην επίλυση των πραγματικών προβλημάτων αποκαλύπτει διάφορους περιορισμούς και ανεπάρκειες των τρεχόντων εργαλείων.

Δεδομένου ότι πολλά προβλήματα που λύνονται από τον προγραμματισμό περιορισμών ανήκουν στον τομέα των NP-hard προβλημάτων, ο προσδιορισμός των περιορισμών που καθιστούν το πρόβλημα ανιχνεύσιμο είναι πολύ σημαντικός και από θεωρητικής και πρακτικής άποψης. Εντούτοις, όπως με τις περισσότερες προσεγγίσεις στα NP-hard προβλήματα, η αποδοτικότητα των προγραμμάτων περιορισμού είναι ακόμα απρόβλεπτη και η διαίσθηση είναι συνήθως το πιο σημαντικό μέρος της απόφασης του πότε και πώς πρέπει να χρησιμοποιηθούν οι περιορισμοί. Το πιο κοινό πρόβλημα που αναφέρεται από τους χρήστες των συστημάτων περιορισμών είναι η σταθερότητα του μοντέλου των περιορισμών. Ακόμη και μικρές αλλαγές στο πρόγραμμα ή στα στοιχεία μπορούν να οδηγήσουν σε μια δραματική αλλαγή στην απόδοση. Δυστυχώς, η διαδικασία της αποσφαλμάτωσης για μια σταθερή εκτέλεση σε μια ποικίλα δεδομένων εισόδου, αυτήν την περίοδο δεν γίνεται καλά κατανοητή.

Άλλο ένα πρόβλημα είναι η επιλογή της σωστής τεχνικής ικανοποίησης περιορισμών για κάθε συγκεκριμένο πρόβλημα. Ορισμένες φορές, η γρήγορη τυφλή αναζήτηση, όπως το χρονολογικό Backtracking, είναι πιο αποδοτική από την πιο ακριβή διάδοση περιορισμών και αντίθετα.

Ένα ιδιαίτερο πρόβλημα σε πολλά μοντέλα περιορισμού είναι η βελτιστοποίηση του κόστους. Μερικές φορές, είναι πολύ δύσκολο να βελτιωθεί μια αρχική λύση, και μια μικρή βελτίωση παίρνει πολύ περισσότερο χρόνο από ότι να βρεθεί η αρχική λύση. Υπάρχει μια ανταλλαγή μεταξύ μιας «οποιαδήποτε» λύσης και της «καλύτερης» λύσης.

Τα προγράμματα περιορισμών είναι επαυξητικά υπό κάποια έννοια (μπορούν να προσθέσουν περιορισμούς δυναμικά) αλλά δεν έχουν καμία υποστήριξη για απευθείας λύση online που απαιτείται στο τρέχον μεταβαλλόμενο περιβάλλον. Τις

περισσότερες φορές, τα συστήματα περιορισμών παράγουν σχέδια που έπειτα εκτελούνται, αλλά οι μηχανές παθαίνουν βλάβες, τα αεροπλάνα καθυστερούν και οι νέες διαταγές έρχονται στη χειρότερη χρονική στιγμή. Αυτό απαιτεί γρήγορο επανασχεδιασμό ή βελτίωση της τρέχουσας λύσης για να απορροφούν τα απροσδόκητα γεγονότα. Πάλι, υπάρχει ανταλλαγή μεταξύ της βελτιστοποίησης της λύσης, που σημαίνει συνήθως το σφιχτό πρόγραμμα, και τη λιγότερο βέλτιστης αλλά σταθερής λύση που απορροφά τις μικρές αποκλίσεις.

2.8 Τάσεις.

Οι ανεπάρκειες των τρεχόντων συστημάτων ικανοποίησης περιορισμών χαρακτηρίζουν τις κατευθύνσεις για την περαιτέρω ανάπτυξη. Μεταξύ τους, η μοντελοποίηση φαίνεται μια από τις σημαντικότερες. Οι συζητήσεις άρχισαν για τη χρησιμοποίηση των καθολικών περιορισμών που ενσωματώνουν τους πρωτόγονους περιορισμούς σε μια αποδοτικότερη συσκευασία. Μια γενικότερη ερώτηση αφορά τις γλώσσες μοντελοποίησης για να εκφραστούν τα προβλήματα περιορισμών. Αυτήν την περίοδο, τα περισσότερα πακέτα ικανοποίησης περιορισμών είναι είτε επεκτάσεις μιας γλώσσας προγραμματισμού (CLP) είτε βιβλιοθήκες που χρησιμοποιούνται με τις συμβατικές γλώσσες προγραμματισμού (Solver ILOG)[24]. Γλώσσες διαμόρφωσης περιορισμών παρόμοιες με την αλγεβρική περιγραφή εισάγονται για να απλοποιήσουν την περιγραφή των περιορισμών (Numerica)[25] ή οπτικές γλώσσες διαμόρφωσης χρησιμοποιούνται για να παράγουν προγράμματα περιορισμών από οπτικά σχέδια (VisOpt VML)[26].

Από τη χαμηλότερου επιπέδου άποψη, οι τεχνικές απεικόνισης για την αναζήτηση κατανόησης (*understanding search*) γίνονται δημοφιλέστερες δεδομένου ότι βοηθούν να προσδιοριστούν οι δυσχέρειες του συστήματος. Η αναζήτηση ελέγχου (*controlling search*) είναι πιθανώς ένα από τα λιγότερο αναπτυγμένα μέρη του προγραμματισμού περιορισμών.

Η μελέτη των αλληλεπιδράσεων των διάφορων μεθόδων επίλυσης περιορισμών είναι από τα πιο προκλητικά προβλήματα. Οι υβριδικοί αλγόριθμοι που συνδυάζουν διάφορες τεχνικές επίλυσης περιορισμών είναι ένα από τα αποτελέσματα αυτής της έρευνας.

Τέλος ένας άλλος ενδιαφέρων τομέας μελέτης είναι το *solver collaboration* και ο συσχετικός συνδυασμός θεωριών (*correlative combination of theories*). Ο συνδυασμός τεχνικών ικανοποίησης περιορισμών με παραδοσιακές μεθόδους επιχειρησιακής έρευνας όπως τον προγραμματισμό ακέραιων αριθμών είναι μια άλλη πρόκληση της τρέχουσας έρευνας.

Κεφάλαιο 3

Μοντελοποίηση προβλήματος

3.1 Γενικά

Η μοντελοποίηση ενός προβλήματα χρονοπρογραμματισμού αποτελεί ένα από τα βασικά μέρη της λύσης του κάθε προβλήματος. Με την μοντελοποίηση του εκάστοτε προβλήματος timetabling εννοούμε τον ορισμό των μεταβλητών, των πεδίων τιμών τους και των περιορισμών. Βασικός σκοπός της διαδικασίας μοντελοποίησης είναι να προσπαθεί να προσεγγίσει όσο το δυνατόν περισσότερο την πραγματικότητα. Μ' αυτό τον τρόπο διευκολύνεται ο υπολογιστικός προγραμματισμός του προβλήματος καθώς και η ποιότητα των λύσεων που θα παραχθούν, αφού θα προσομοιωθεί, όσο γίνεται, η πραγματικότητα. Επομένως, οι μοντελοποιήσεις που μπορεί να έχει ένα πρόβλημα ποικίλουν ανάλογα με τον πόσο θέλει ο προγραμματιστής να προσομοιώσει την πραγματικότητα και πόσο λεπτομερείς και ακριβείς θέλει να είναι οι λύσεις του. Όμως, όσο πιο λεπτομερής είναι η μοντελοποίηση και όσο πιο πολύ θέλουμε να προσομοιωθεί η πραγματικότητα, τόσο πιο πολύπλοκο θα είναι η κωδικοποίηση του προβλήματος σε μια γλώσσα προγραμματισμού από τον προγραμματιστή. Έτσι βλέπουμε συχνά, να γίνονται κάποιες παραδοχές κατά τη διάρκεια της κωδικοποίησης για να διευκολυνθεί σε ένα βαθμό η διαδικασία.

Παρακάτω θα γίνει, αρχικά, μια περιγραφή του προβλήματος και θα δοθεί η δομή του. Στη συνέχεια, θα αναλυθούν οι δομές δεδομένων που θα χρησιμοποιηθούν για να αναπαρασταθεί το πρόβλημα. Τελικά, θα σχολιαστούν οι περιορισμοί που θεωρήθηκαν απαραίτητοι και χρησιμοποιήθηκαν για το παρών πρόβλημα.

3.2 Μοντελοποίηση προβλήματος κατασκευής προγράμματος εξεταστικής περιόδου.

Το πρόβλημα κατασκευής προγράμματος μπορεί να μοντελοποιηθεί με πολλούς τρόπους. Η διαφορά τους έγκειται στο χαρακτήρα των μεταβλητών που θα χρησιμοποιηθούν και στο πεδίο τιμών τους.

Ως πρόβλημα της κατασκευής του ωρολογίου προγράμματος εξεταστικής περιόδου ορίζεται η τοποθέτηση των μαθημάτων όλων των ακαδημαϊκών εξαμήνων σε ένα δοσμένο εύρος ημερών, ωρών και αιθουσών με διάφορους περιορισμούς να επηρεάζουν τη θέση του καθενός.

Θεωρήθηκε ότι κάθε μέρα της εβδομάδας αποτελείται από τέσσερα ακαδημαϊκά τρίωρα (9:00-12:00, 12:00-15:00, 15:00-18:00, 18:00-21:00), τα οποία δεν μπορούν να διασπαστούν περεταίρω και καθένα από αυτά αντιστοιχεί στην εξέταση ενός μαθήματος. Επίσης, υπάρχουν συνολικά δεκαπέντε διαθέσιμες μέρες (τρεις εβδομάδες) και πέντε διαθέσιμες αίθουσες εξέτασης. Κάθε μάθημα διδάσκεται από ένα μόνο καθηγητή και, εκτός από τον καθηγητή, χαρακτηρίζεται από το εξάμηνο στο οποίο ανήκει. Η παράμετρος του εξαμήνου χρησιμεύει καθώς βοηθάει στο να μην

εξετάζονται ποτέ μαθήματα του ίδιου εξαμήνου την ίδια μέρα. Φυσικά, δεν μπορούν να εξεταστούν δύο διαφορετικά μαθήματα την ίδια μέρα, ώρα και στην ίδια αίθουσα. Δεν μπορούν να εξεταστούν ταυτόχρονα δύο μαθήματα του ίδιου καθηγητή. Τα μαθήματα του ίδιου εξαμήνου δεν θα πρέπει να συνωστίζονται, αλλά να υπάρχουν όσο το δυνατό, μέρες κενές από μάθημα σε μάθημα. Τέλος θα πρέπει να εξετάζονται τουλάχιστον δύο και το πολύ τέσσερα μαθήματα κάθε μέρα, έτσι ώστε να αποφευχθούν μέρες στις οποίες δεν εξετάζεται κανένα μάθημα και μέρες που εξετάζονται υπερβολικά πολλά μαθήματα μαζί αντίστοιχα.

Κατά τη διάρκεια της προετοιμασίας βρέθηκαν δύο πιθανοί τρόποι μοντελοποίησης του συγκεκριμένου προβλήματος. Η πρώτη περίπτωση είναι να έχουμε σταθερές μέρες και ώρες και αίθουσες εξέτασης, στις οποίες θα αναθέτονται μαθήματα και αντίστοιχα ο καθηγητής και το εξάμηνο του κάθε μαθήματος. Στην περίπτωση αυτή, τα αντικείμενα που θα τοποθετούνται θα είναι τα μαθήματα. Στη δεύτερη περίπτωση, τα μαθήματα θα είναι σταθερά, με το καθένα να έχει πάλι τον καθηγητή του και το εξάμηνο στο οποίο διδάσκεται, στα οποία θα ανατίθεται μια μέρα, μια ώρα και μια αίθουσα εξέτασης. Άρα τα αντικείμενα που θα αναθέτονται σε χρονοθυρίδες θα είναι οι τρεις μεταβλητές που αναφέρθηκαν. Επιλέχθηκε η δεύτερη περίπτωση ως η ιδανική για το πρόβλημά μας αφού αντιπροσωπεύει τον τρόπο που επιλύεται το συγκεκριμένο πρόβλημα στην πραγματική ζωή. Επιπλέον λόγοι για τους οποίους επιλέχθηκε ο δεύτερος τρόπος μοντελοποίησης είναι η απλότητά του στην υλοποίηση και η καλύτερή του απόδοση, αφού στην πρώτη περίπτωση θα έπρεπε να ανατεθούν τέσσερις μεταβλητές κάθε φορά ενώ στη δεύτερη μόνο τρεις.

3.3 Δομές δεδομένων.

3.3.1. Γενικά

Επειδή η δομή του προβλήματος το καθιστά σύνθετο, παρουσιάστηκε, αρχικά, ένα δίλημμα σχετικά με την τεχνική που θα ακολουθηθεί για την υλοποίησή του και με το ποια θα είναι η βασική του δομή. Οι δύο τεχνικές που εξετάστηκαν ήταν η συναρτησιακή (με χρήση συναρτήσεων) και ο αντικειμενοστραφής προγραμματισμός. Το όφελος της συναρτησιακής μεθόδου είναι η απλότητά της. Από την άλλη όμως, τα θετικά του αντικειμενοστραφούς προγραμματισμού είναι πολλά και επειδή επιλέχθηκε η χρήση μιας αντικειμενοστραφούς γλώσσας, τελικά οδηγηθήκαμε στη χρήση της συγκεκριμένης μεθόδου. Το γεγονός αυτό επηρέασε πολύ τις δομές δεδομένων που θα χρησιμοποιήσουμε, όπως θα δειχθεί παρακάτω.

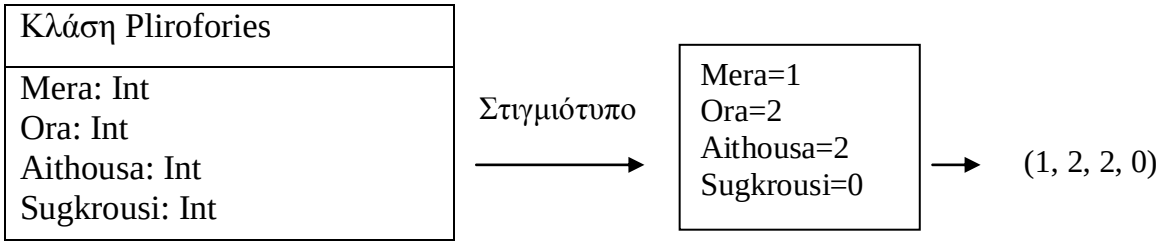
3.3.2. Δομές δεδομένων

Η σύνθεση του προβλήματος κατέστησε αδύνατο να γνωρίζουμε από την αρχή ποια θα είναι η τελική δομή που θα μπορεί να υποστηρίξει της πληροφορίες που θέλουμε να κρατήσουμε για κάθε μάθημα. Το πρόβλημα που έπρεπε να αντιμετωπιστεί είναι πως η μεταβλητή του προβλήματός μας είναι πολυδιάστατη αφού τα τρία μέρη της (μέρα, ώρα και αίθουσα εξέτασης) θα πρέπει να αλλάζουν συνεχώς. Τελικά αποφασίστηκε να χρησιμοποιηθούν δύο κλάσεις οι οποίες θα

κρατούν όλες τις διαθέσιμες πληροφορίες. Οι πληροφορίες στις οποίες αναφερόμαστε είναι οι εξής:

- Μέρα εξέτασης
- Ώρα εξέτασης
- Αίθουσα εξέτασης
- Όνομα καθηγητή μαθήματος
- Εξάμηνο διδασκαλίας μαθήματος
- Αν το μάθημα εμπλέκεται σε σύγκρουση ή όχι

Η πρώτη κλάση ονομάστηκε ***Plirofories*** και σ' αυτή θα αποθηκεύονται η μέρα, η ώρα, η αίθουσα εξέτασης του μαθήματος και αν αυτό εμπλέκεται σε σύγκρουση. Η μορφή της κλάσης Plirofories θα έχει την παρακάτω μορφή:

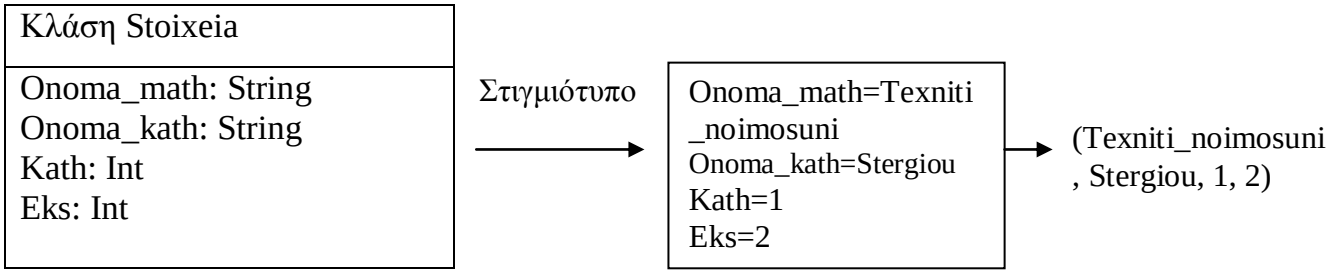


Στο παραπάνω παράδειγμα βλέπουμε κάποιο πιθανό στιγμιότυπο των ιδιοτήτων κάποιου μαθήματος. Οι τιμές αυτές δίνονται από τον αλγόριθμο του προβλήματος κατά την εκτέλεση του προγράμματος και όχι από το χρήστη. Οι ιδιότητες αυτές έχουν συγκεκριμένα πεδία τιμών:

- **Mera**: παίρνει τιμές από το 0 έως 14, αφού έχουμε ορίσει ότι οι μέρες εξέτασης θα είναι δεκαπέντε.
- **Ora**: παίρνει τιμές από 0 έως 3, αφού ορίσαμε τέσσερα ακαδημαϊκά τρίωρα ανά ημέρα.
- **Aithousa**: παίρνει τιμές από 0 έως 4, αφού ορίσαμε πέντε διαθέσιμες αίθουσες εξέτασης.
- **Sugkrousi**: παίρνει τιμή 0 ή 1 ανάλογα με το αν το συγκεκριμένο μάθημα εμπλέκεται σε σύγκρουση.

Στο συγκεκριμένο στιγμιότυπο βλέπουμε πως το συγκεκριμένο μάθημα θα εξεταστεί τη μέρα 1 (Τρίτη της πρώτης εβδομάδας) στο τρίωρο 2 (12:00-15:00) και στην αίθουσα 2. Τέλος, το μάθημα αυτό δεν εμπλέκεται δε καμία σύγκρουση (η συγκεκριμένη ανάθεση τιμών δεν παραβιάζει κανένα περιορισμό).

Η δεύτερη κλάση ονομάστηκα ***Stoixeia*** και σ' αυτή αποθηκεύονται το όνομα του μαθήματος, το όνομα του καθηγητή, ο κωδικός του καθηγητή που διδάσκει το μάθημα και το εξάμηνο κατά το οποίο διδάσκεται το μάθημα. Η μορφή της κλάσης Stoixeia έχει την παρακάτω μορφή:



Το παραπάνω παράδειγμα μας δείχνει τα στοιχεία κάπου μαθήματος. Οι συγκεκριμένες τιμές δίνονται από τον χρήστη σε ένα αρχείο .txt το οποίο και διαβάζει το πρόγραμμα στα αρχικά βήματα της εκτέλεσής του ως είσοδο. Τα πεδία τιμών των παραπάνω μεταβλητών είναι τα παρακάτω:

- **Onoma_math**: είναι συμβολοσειρά οπότε μπορεί να αποτελείται από κάθε συνδυασμό χαρακτήρων.
- **Onoma_kath**: είναι συμβολοσειρά επίσης.
- **Kath**: παίρνει ακέραιες θετικές τιμές που αντιστοιχούν στον κωδικό κάθε καθηγητή. Για ευκολία στην υλοποίηση του προγράμματος επιλέξαμε στο κώδικα να αναφερόμαστε στους καθηγητές με βάση το μοναδικό κωδικό του καθενός, αφού η χρήση και η επεξεργασία αριθμών είναι πολύ πιο εύκολη από την αντίστοιχη των συμβολοσειρών.
- **Eks**: παίρνει επίσης ακέραιες θετικές τιμές.

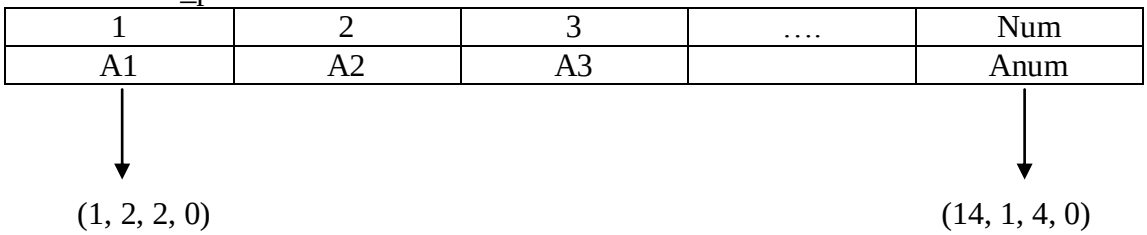
Το συγκεκριμένο στιγμιότυπο μας δείχνει πως το μάθημα Τεχνητή Νοημοσύνη διδάσκεται από τον καθηγητή Στεργίου με κωδικό 1 στο εξάμηνο 2.

Οι προηγούμενες δομές δεδομένων έχουν άμεση σχέση με την επόμενη. Τα αντικείμενα των κλάσεων Plirofories και Stoixeia θα εισάγονται σε δύο πίνακες γραμμές οι οποίοι ονομάζονται ***Arhikos_pinakas*** και ***Lista*** και θα έχουν τύπους δεδομένων Plirofories και Stoixeia αντίστοιχα. Δηλαδή:

- Arhikos_pinakas = new Plirofories[num];
- Lista = new Stoixeia[num];

Το μέγεθος των παραπάνω πινάκων θα ισούται με τον συνολικό αριθμό των μαθημάτων. Διαγραμματικά, η μορφή τους θα είναι η παρακάτω:

Arhikos_pinakas:



Οι τιμές των κελιών του παραπάνω πίνακα θα αλλάζουν κατά τη διάρκεια της εκτέλεσης του προγράμματος ανάλογα με τους εκάστοτε περιορισμούς. Στο τέλος της εκτέλεσης, οι τιμές που θα βρίσκονται στο πίνακα θα αντιστοιχούν στη λύση που επέστρεψε ο αλγόριθμος και θα αντιπροσωπεύουν το τελικό ωρολόγιο πρόγραμμα εξεταστικής.

Stoixeia:

1	2	3	….	Num
S1	S2	S3		Snum

(Texniti_noimosuni, Stergiou, 5, 3) (Ilektroniki_epistimi, Dimitropoulos, 18, 5)

Για να βρει λύση κάποιος αλγόριθμος, συνήθως, χρησιμοποιείται κάποιο κριτήριο λύσης. Στη δική μας περίπτωση το κριτήριο αυτό θα είναι ο αριθμός των συνολικών συγκρούσεων. Έτσι, επιλέχθηκε να χρησιμοποιηθεί μια μεταβλητή με όνομα *count_sugkrousewn* η οποία θα κρατάει ανά πάσα στιγμή την τρέχουσα τιμή του συνολικού αριθμού συγκρούσεων. Στόχος του αλγορίθμου και ταυτόχρονα κριτήριο λύσης θα είναι ο μηδενισμός την τιμή της count_sugkrousewn με διαδοχικές διαφορετικές αναθέσεις τιμών σε μεταβλητές ώστε στο τέλος της εκτέλεσής του να μην παρουσιάζεται καμία σύγκρουση. Για να προκύψει η τιμή της count_sugkrousewn, μετά από κάθε ανάθεση τιμών, γίνεται έλεγχος των περιορισμών που θα αναλυθούν παρακάτω ώστε να βρεθεί ποιοι παραβιάζονται και ποιες τιμές των μεταβλητών είναι υπεύθυνες για την κάθε παραβίαση. Μόλις βρεθεί κάποιος περιορισμός που παραβιάζεται και εντοπιστεί η τιμή που τον παραβιάζει, η count_sugkrousewn αυξάνεται κατά ένα και η τιμή της μεταβλητής *sugkrousi* για τη συγκεκριμένη μεταβλητή στον πίνακα Arhikos_pinakas γίνεται ίση με 1 (αρχικά έχουν αρχικοποιηθεί όλες στο 0). Έτσι το πρόγραμμα γνωρίζει κάθε στιγμή πόσες συγκρούσεις υπάρχουν και ποιες μεταβλητές είναι υπεύθυνες για αυτές, ώστε να τις αλλάξει διαδοχικά μέχρι να μην παραβιάζεται κανένας περιορισμός.

3.4 Περιορισμοί.

Οι περιορισμοί του προβλήματος έχουν αναφερθεί στην αρχή του κεφαλαίου αλλά θα γίνει μια περαιτέρω ανάλυση τους .Συγκεκριμένα ,για κάθε περιορισμό θα δίνεται μία περιγραφή και ένα γενικό παράδειγμα . Όλοι οι περιορισμοί που θα δούμε είναι κυρίως μεγαλύτερης τάξεως, γιατί όπως θα φανεί στη συνέχεια εμπλέκονται αρκετές ιδιότητες του εκάστοτε μαθήματος κάθε φορά. Ο αλγόριθμος λειτουργεί ελέγχοντας τις τιμές των ιδιοτήτων σε ένα ζευγάρι μαθημάτων που έχουν επιλεχθεί. Όλοι οι περιορισμοί έχουν μορφή ελέγχου με διαδοχικά if. Αν η συνθήκες των if δεν ικανοποιούνται τότε το πρόγραμμα θα προχωράει παρακάτω αφού δεν θα βρίσκει σύγκρουση. Αντίθετα, αν ικανοποιούνται οι συνθήκες, τότε θα βρίσκεται σύγκρουση και θα εκτελούνται οι εκάστοτε εντολές. Έστω ότι τα μαθήματα που έχει επιλεγεί να ελεγχθούν οι ιδιότητές τους είναι τα *i* και *j*. Οι περιορισμοί που τέθηκαν στο πρόβλημα όπως αυτό μοντελοποιήθηκε είναι οι εξής :

- Μαθήματα του ίδιου εξαμήνου δεν εξετάζονται την ίδια μέρα. Υλοποιείται ως εξής :

```
if (Arhikos_pinakas[i].get_mera() == Arhikos_pinakas[j].get_mera()) {
  if (Lista[i].get_eks() == Lista[j].get_eks()) {
  }
}
```

- Δύο διαφορετικά μαθήματα δεν μπορούν να εξεταστούν την ίδια μέρα, την ίσια ώρα και στην ίδια αίθουσα. Ένας απόλυτα φυσιολογικός περιορισμός αφού είναι πρακτικά αδύνατο να εξετάζονται διαφορετικά μαθήματα ταυτόχρονα. Υλοποιείται ως εξής:

```
if (Arhikos_pinakas[i].get_mera() == Arhikos_pinakas[j].get_mera()) {
  if (Arhikos_pinakas[i].get_ora() == Arhikos_pinakas[j].get_ora()) {
    if (Arhikos_pinakas[i].get_aithousa() == Arhikos_pinakas[j].get_aithousa()) {
    }
  }
}
```

- Δυο μαθήματα του ίδιου καθηγητή δεν μπορούν να εξετάζονται την ίδια ώρα, κάτι πολύ λογικό αφού το ίδιο πρόσωπο δεν μπορεί να βρίσκεται σε δύο μέρη ταυτόχρονα. Υλοποιείται ως εξής:

```
if (Arhikos_pinakas[i].get_mera() == Arhikos_pinakas[j].get_mera()) {
  if (Arhikos_pinakas[i].get_ora() == Arhikos_pinakas[j].get_ora()) {
    if (Lista[i].get_kodiko_kath() == Lista[j].get_kodiko_kath()) {
    }
  }
}
```

- Για να αποφύγουμε την εξέταση μαθημάτων του ίδιου εξαμήνου σε διάστημα πολύ λίγων ημερών (π.χ. και τα πέντε μαθήματα κάποιου εξαμήνου να εξετάζονται όλα σε μία εβδομάδα) προσθέσαμε το παρακάτω περιορισμό. Ελέγχει αν οι μέρες εξέτασης των δύο μαθημάτων είναι διαδοχικές. Αν ναι, αυξάνει τη τιμή ενός μετρητή κατά ένα. Αν η τιμή αυτού του μετρητή υπερβεί το τρία τότε βρίσκει σύγκρουση. Έτσι θα μπορούμε να έχουμε το πολύ μέχρι τρία μαθήματα που εξετάζονται διαδοχικές μέρες. Υλοποιείται ως εξής:

```
if (Lista[i].get_eks() == Lista[j].get_eks()) {
  if ((Arhikos_pinakas[i].get_mera() - Arhikos_pinakas[j].get_mera() == 1) ||
(Arhikos_pinakas[j].get_mera() - Arhikos_pinakas[i].get_mera() == 1)) {
    count_apostasewn[Lista[i].get_eks()]++;
    if (count_apostasewn[Lista[i].get_eks()] > 3) {
    }
  }
}
```

- Για να αποφύγουμε μέρες που εξετάζονται πολύ λίγα μαθήματα (ακόμα και κανένα) προσθέσαμε ένα περιορισμό που εξασφαλίζει ότι θα εξετάζονται τουλάχιστον δύο μαθήματα κάθε μέρα. Χρησιμοποιεί ένα πίνακα μετρητή με όνομα *count_mathimaton* και μέγεθος ίσο με τις διαθέσιμες μέρες εξέτασης (15 στη περίπτωσή μας) ο οποίος κρατά τον αριθμό των μαθημάτων που εξετάζονται στη κάθε μέρα. Υλοποιείται ως εξής:

```
if (count_mathimaton[i] < 2) {
}
```

- Τέλος, αντίστοιχα, για να αποφύγουμε μέρες που εξετάζονται πάρα πολλά μαθήματα προσθέσαμε ένα περιορισμό που εξασφαλίζει ότι θα εξετάζονται το πολύ τέσσερα μαθήματα κάθε μέρα. Υλοποιείται ως εξής:

```
if (count_mathimaton[i] > 4) {
}
```

Κεφάλαιο 4

Περιγραφή αλγορίθμων

4.1.Γενικά

Στο κεφάλαιο αυτό θα αναλυθεί ο αλγόριθμος που επιλέχθηκε και αναπτύχθηκε για την υλοποίηση της κατασκευής ωρολογίου προγράμματος εξεταστικής περιόδου. Θα δοθεί μια λεπτομερής περιγραφή, ο ψευδοκώδικάς του και ο τρόπος που διαχειρίζεται τις δομές δεδομένων στο πρόβλημά μας. Ο αλγόριθμος που επιλέχτηκε, εκτός του ότι είναι αλγόριθμος τοπικής αναζήτησης, έχει το χαρακτηριστικό ότι ανήκει στη κατηγορία των ευρεστικών, χρησιμοποιεί δηλαδή κάποια πληροφορία ως ένδειξη για να φτάσει στη λύση του προβλήματος. Η ένδειξη αυτή δεν δείχνει τη λύση του προβλήματος, αλλά το «δρόμο» προς τη λύση.

Ο αλγόριθμος που επιλέξαμε είναι ο αλγόριθμος ελάχιστων συγκρούσεων με τυχαίες επανεκκινήσεις (min-conflicts with random restarts).

4.2. Ο αλγόριθμος ελάχιστων συγκρούσεων με τυχαίες επανεκκινήσεις (min-conflicts with random restarts)

4.2.1.Εισαγωγή

Ο αλγόριθμος ελάχιστων συγκρούσεων με τυχαίες επανεκκινήσεις, ή απλά ο min-conflicts with random restarts (για ευκολία θα χρησιμοποιείται το όνομα Random-Restarts για τον αλγόριθμο αυτό), χρησιμοποιείται ευρύτατα σε προβλήματα ικανοποίησης περιορισμών και ειδικότερα σε προβλήματα χρονοπρογραμματισμού. Προτάθηκε από τον Gu το 1989 και αναπτύχθηκε από τον Minton το 1992. Έχει δειχτεί ότι ο συγκεκριμένος αλγόριθμος μπορεί να εφαρμοστεί και να λύσει εύκολα το πρόβλημα των 1.000.000 βασιλισσών σε μέσο όρο πενήντα βήματα και το πρόβλημα των 3.000.000 βασιλισσών σε λιγότερο από ένα λεπτό. Ο Random-Restarts έχει μία βασική διαφορά με την min-conflicts: μπορεί να κάνει επανεκκίνηση στο πρόβλημα που εξετάζει. Η επιτυχία του συγκεκριμένου αλγορίθμου έγκειται στην απλή του φιλοσοφία και στη μεγάλη ταχύτητα επίλυσης που διαθέτει όπως προείπαμε. Αυτά τον καθιστούν ιδανικό αλγόριθμο για την λύση του προβλήματός μας.

4.2.2.Ανάλυση αλγορίθμου

Ο min-conflicts με τυχαίες επανεκκινήσεις δέχεται σαν είσοδο ένα πρόβλημα ικανοποίησης περιορισμών και επιστρέφει σαν έξοδο την λύση του προβλήματος αυτού ή αποτυχία. Ειδικότερα, οι είσοδοι είναι το πρόβλημα περιορισμών (CSP), ένας μέγιστος αριθμός βημάτων (max_epanalipseis) με το πέρας του οποίου ο αλγόριθμος θα σταματάει την αναζήτηση λύσης και ένας μέγιστος αριθμός επανεκκινήσεων που επιτρέπεται να τρέξει ο αλγόριθμος πριν παραιτηθεί από την προσπάθεια για επίλυση και θα επιστρέφει αποτυχία. Τα βήματα του αλγορίθμου αναλυτικά είναι τα παρακάτω:

1. Γίνεται μια τυχαία ανάθεση τιμών σε όλες τις μεταβλητές του προβλήματος.
2. Γίνεται έλεγχος συγκρούσεων και καταγράφονται τα μαθήματα που εμπλέκονται σε σύγκρουση και ο συνολικός αριθμός συγκρούσεων.
3. Ο αλγόριθμος μπαίνει σε ένα βρόγχο με όριο βημάτων τον αριθμό των μέγιστων επαναλήψεων.
4. Εάν η ανάθεση τιμών που έγινε αποτελεί λύση (δεν υπάρχουν συγκρούσεις) επιστρέφεται η ανάθεση αυτή και τερματίζει ο αλγόριθμος.
5. Εάν η αρχική ανάθεση δεν αποτελεί λύση, τότε επιλέγεται μία από τις μεταβλητές (μαθήματα) που εμπλέκονται σε σύγκρουση.
6. Δίνονται διαδοχικά σ’ αυτή τη μεταβλητή όλες οι πιθανές τιμές από το πεδίο ορισμού της και ξαναγίνεται έλεγχος συγκρούσεων κάθε φορά.
7. Στο τέλος όλων των παραπάνω αναθέσεων δίνεται στη μεταβλητή η τιμή που μειώνει τον αριθμό των συγκρούσεων περισσότερο από τις υπόλοιπες.
8. Η ίδια διαδικασία (βήματα 4 έως 7) επαναλαμβάνεται έως ότου δεν υπάρχουν άλλες συγκρούσεις ή έχει συμπληρωθεί ο μέγιστος αριθμός επαναλήψεων.
9. Εάν δεν έχει βρεθεί λύση ο αλγόριθμος ξεκινά από την αρχή με μια νέα τυχαία ανάθεση τιμών σε όλες τις μεταβλητές και εκτελεί ξανά τα παραπάνω βήματα (1-8). Η ίδια διαδικασία επαναλαμβάνεται έως ότου ο αλγόριθμος να βρει λύση ή να συμπληρωθεί ο μέγιστος αριθμός επανεκκινήσεων.
10. Εάν έχει βρεθεί λύση τότε ο αλγόριθμος επιστρέφει την πλήρη ανάθεση τιμών που δίνει λύση αλλιώς επιστρέφει αποτυχία.

Ο ψευδοκώδικας του αλγορίθμου min-conflicts που περιγράφει πλήρως την λειτουργία του αλγορίθμου είναι ο παρακάτω:

Function RANDOM-RESTARTS(csp, max_epanalipseis, max_epanekkiniseis) επιστρέφει μία λύση ή αποτυχία.

Είσοδοι : csp, ένα πρόβλημα ικανοποίησης περιορισμών
max_epanalipseis, μέγιστος αριθμός βημάτων
max_epanekkiniseis, μέγιστος αριθμός επανεκκινήσεων πριν εγκαταλείψει ο αλγόριθμος

for j=1 to max_epanekkiniseis **do**
current ← μία πλήρης ανάθεση τιμών στο csp
for i=1 to max_epanalipseis **do**
if current είναι μία λύση του csp ,**τότε επέστρεψε** το current
var←μία τυχαία επιλεγμένη μεταβλητή ,που έχει conflicts από το VARIABLES[CSP]
value←μία τιμή v για την μεταβλητή var που ελαχιστοποιεί την CONFLICTS(var, v, current, CSP)
var = value στο current
επέστρεψε αποτυχία

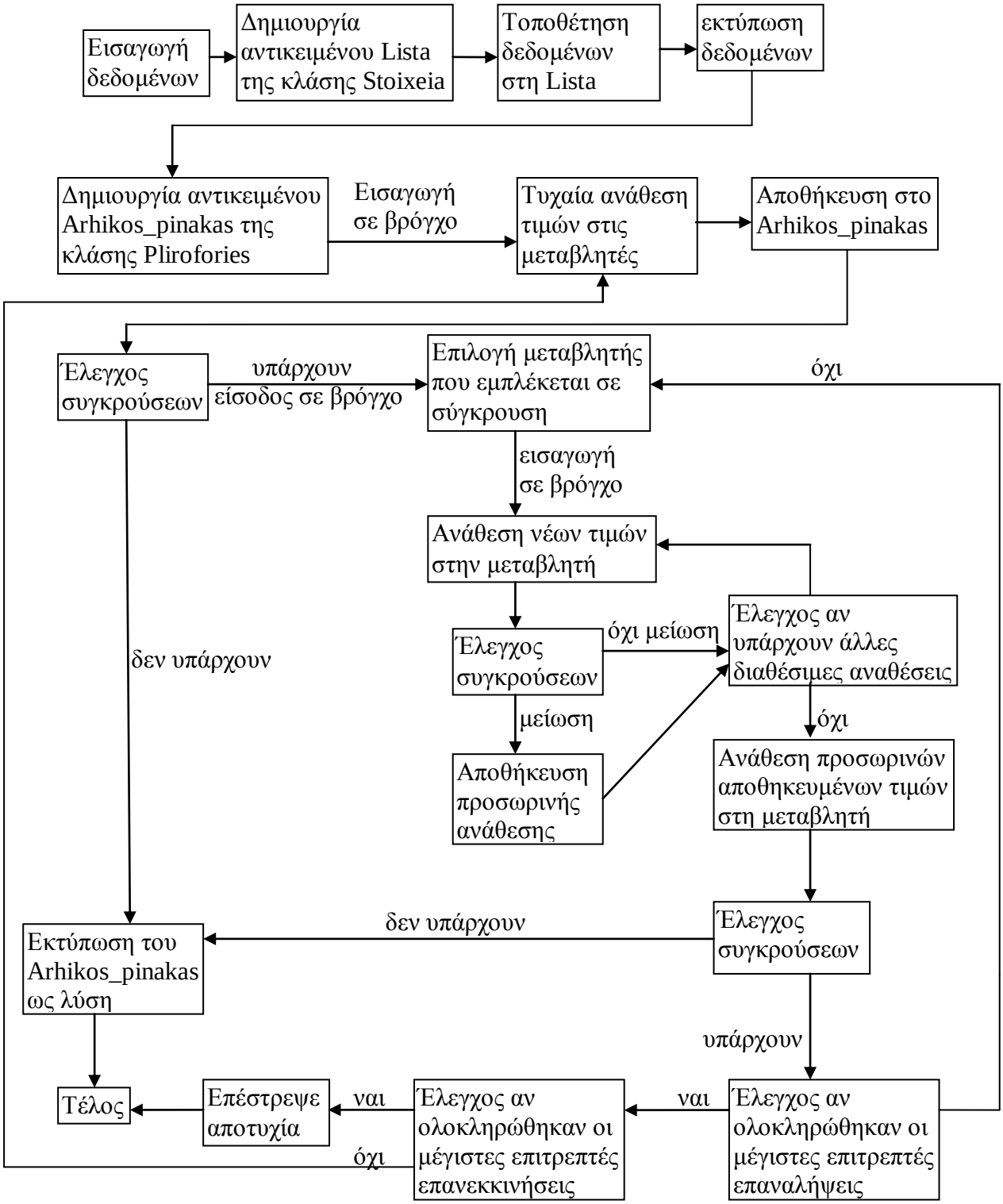
Το σύνολο VARIABLES[CSP] είναι το σύνολο των μεταβλητών του προβλήματος προς επίλυση. Η συνάρτηση CONFLICTS(var, v, current, CSP) μετρά τον αριθμό των περιορισμών που παραβιάζονται από μία ανάθεση τιμής σε μια συγκεκριμένη μεταβλητή, αφού συγκριθεί με τις υπόλοιπες μεταβλητές μίας πλήρους ανάθεσης.

Όπως βλέπουμε, η παραπάνω γενική περιγραφή του αλγορίθμου Random-Restarts δεν είναι αρκετή για να αναλύσει πλήρως τη λειτουργία του. Για αυτό το λόγο κρίθηκε απαραίτητο να δοθεί και μια περιγραφή του τρόπου με τον οποίο ο αλγόριθμος διαχειρίζεται τις δομές δεδομένων του προβλήματός μας.

4.2.3.Διαχείριση δομών δεδομένων από min-conflicts με τυχαίες επανεκκινήσεις

Η αλληλεπίδραση του αλγορίθμου με τις δομές δεδομένων παρουσιάζει μεγάλο ενδιαφέρον γιατί δείχνει ,όχι μόνο πως διαχειρίζονται οι δομές δεδομένων, αλλά και την κίνηση του αλγόριθμου κατά τη διάρκεια της προσπάθειας εύρεσης λύσης. Υπενθυμίζεται ότι οι υπάρχουσες δομές που χρησιμοποιήθηκαν είναι Plirofories, Stoixeia, Lista και Arhikos_pinakas.

Διαγραμματικά ο αλγόριθμος φαίνεται στο σχήμα που ακολουθεί :



Πιο αναλυτικά, το παραπάνω διάγραμμα του αλγορίθμου Random-Restarts επεξηγείται παρακάτω:

Ορίζεται από το χρήστη το πρόβλημα και εισάγονται τα δεδομένα εισόδου. Ως δεδομένα εισόδου εννοούμε τα στοιχεία του κάθε μαθήματος, το όνομά του, το όνομα του καθηγητή που το διδάσκει, τον κωδικό του καθηγητή και το εξάμηνο κατά το οποίο διδάσκεται το μάθημα. Τα στοιχεία αυτά δίνονται στο πρόγραμμα σε ένα αρχείο κειμένου και έχουν τη παρακάτω μορφή:

Δίκτυα 1	Λάγκας	9	2
Ηλεκτρικά κυκλώματα	Θεοδουλίδης	10	2
Βάσεις δεδομένων	Συμεωνίδης	14	3
Τεχνητή όραση	Δημητρόπουλος	18	5
Θεωρία πολυπλοκότητας	Τσαλικάκης	27	5

Στη συνέχεια, τα στοιχεία του κάθε μαθήματος τοποθετούνται στη δομή Lista όπου κάθε γραμμή αντιστοιχεί σε ένα μάθημα. Δηλαδή, για τον παραπάνω πίνακα το πρώτο μάθημα θα έχει όνομα Δίκτυα 1, θα διδάσκεται από τον καθηγητή Λάγκα με κωδικό 9 και θα είναι μάθημα του $2^{ου}$ εξαμήνου. Παρόμοια θα αρχικοποιηθούν και τα υπόλοιπα μαθήματα. Μετά την ολοκλήρωση της αρχικοποίησης όλων των μαθημάτων εκτυπώνεται η λίστα με τα συνολικά στοιχεία των συνολικών μαθημάτων προς εξέταση που θα έχει τη παρακάτω μορφή:

Onoma_math	Onoma_kath	kath	eks
Δίκτυα 1	Λάγκας	9	2
Ηλεκτρικά κυκλώματα	Θεοδουλίδης	10	2
Βάσεις δεδομένων	Συμεωνίδης	14	3
Τεχνητή όραση	Δημητρόπουλος	18	5
Θεωρία πολυπλοκότητας	Τσαλικάκης	27	5

Στο επόμενο βήμα γίνεται τυχαία ανάθεση τιμών στις υπόλοιπες μεταβλητές που δεν έχουν αρχικοποιηθεί ακόμα. Δίνεται, δηλαδή, για κάθε μάθημα μια τιμή για τη μέρα, την ώρα και την αίθουσα στην οποία θα εξεταστεί. Οι τιμές αυτές δίνονται από μια γεννήτρια τυχαίων αριθμών και κυμαίνονται στο εύρος του πεδίου τιμών κάθε μεταβλητής. Για τις ημέρες οι διαθέσιμες τιμές είναι από 0 έως 14 (3 εβδομάδες εξέτασης), για την ώρα οι διαθέσιμες τιμές είναι από 0 έως 3 (4 τρίωρα κάθε μέρα) και για την αίθουσα εξέτασης οι διαθέσιμες τιμές είναι από 0 μέχρι 4 (5 αίθουσες). Οι παραπάνω τιμές αποθηκεύονται στη δομή Arhikos_pinakas όπου κάθε γραμμή αντιστοιχεί σε ένα μάθημα κατά αντιστοιχία με τη δομή Lista. Έτσι, για παράδειγμα, η δεύτερη γραμμή της δομής Lista και η δεύτερη γραμμή της δομής Arhikos_pinakas θα περιέχουν στοιχεία για το ίδιο μάθημα με κωδικό 2 ($2^{η}$ γραμμή σε κάθε δομή). Έτσι θα μπορούμε να διαβάσουμε και να τροποποιήσουμε όποια στοιχεία του κάθε μαθήματος επιθυμούμε ανεξάρτητα με το αν βρίσκονται σε διαφορετική δομή. Μετά την ολοκλήρωση της τυχαίας αρχικοποίησης η δομή Arhikos_pinakas θα έχει τη παρακάτω μορφή:

mera	ora	aithousa	sugkrousi
3	2	4	0
9	0	2	0
1	1	1	0
14	3	0	0
2	3	3	0

Στη στήλη sugkrousi αρχικά όλες οι τιμές αρχικοποιούνται στο 0, υποθέτοντας ότι κανένα μάθημα δεν εμπλέκεται σε σύγκρουση.

Η επόμενη κίνηση του αλγορίθμου είναι να κάνει έλεγχο συγκρούσεων. Πραγματοποιεί σειριακό έλεγχο όλων των περιορισμών για όλες τις μεταβλητές ώστε να εντοπίσει τον συνολικό αριθμό παραβάσεων περιορισμών (συγκρούσεις) καθώς

και εκείνες τις μεταβλητές οι οποίες εμπλέκονται σε κάποια παραβίαση. Στο τέλος του ελέγχου, η τιμή του πεδίου sugkrousi κάθε μεταβλητής, η οποία εμπλέκεται σε παραβίαση, γίνεται 1 (από 0 που είχαν αρχικοποιηθεί όλες). Ταυτόχρονα αποθηκεύεται σε μια μεταβλητή ο συνολικός αριθμός συγκρούσεων.

Εάν δεν έχουν εντοπιστεί συγκρούσεις, ό αλγόριθμος επιστρέφει την δομή Arhikos_pinakas σαν λύση του προβλήματος, εκτυπώνει τα αποτελέσματα και τερματίζει. Αντίθετα, εάν εντοπίσει κάποιο αριθμό συγκρούσεων, εισέρχεται σε ένα βρόγχο ο οποίος τερματίζει όταν δεν εντοπίζονται πλέον συγκρούσεις ή όταν ολοκληρωθούν οι μέγιστες επιτρεπτές επαναλήψεις (max_epanalipseis). Όσο βρίσκεται στο βρόγχο αυτό, τα βήματα που ακολουθεί είναι τα εξής:

- αρχικά επιλέγει τυχαία κάποια από τις μεταβλητές που εμπλέκονται σε σύγκρουση με βάση την τιμή του πεδίου sugkrousi της καθεμίας.
- Στη συνέχεια εισέρχεται σε ένα νέο βρόγχο στον οποίο σειριακά αναθέτει σ' αυτή τη μεταβλητή όλες τους πιθανούς συνδυασμούς τιμών στα πεδία mera, ora και aithousa. Μετά από κάθε ανάθεση κάνει έλεγχο συγκρούσεων και αν ο αριθμός τους έχει μειωθεί κρατάει την ανάθεση αυτή σε κάποιες προσωρινές μεταβλητές. Η ίδια διαδικασία συνεχίζεται μέχρι να δοθούν όλες οι διαθέσιμες αναθέσεις τιμών στη μεταβλητή και τότε εξέρχεται από τον κόμβο.
- Έπειτα αναθέτει στη μεταβλητή που επέλεξε τις τιμές που έχουν κρατηθεί στις προσωρινές μεταβλητές καθώς για αυτές ο αριθμός των συγκρούσεων έχει μειωθεί περισσότερο.
- Τα παραπάνω βήματα συνεχίζονται έως ότου να ολοκληρωθούν οι μέγιστες επιτρεπτές επαναλήψεις (max_epanalipseis) ή να γίνει ο αριθμός των συγκρούσεων 0. Και στις δύο περιπτώσεις ο αλγόριθμος εξέρχεται από το βρόγχο.

Εάν ο λόγος εξόδου από το βρόγχο είναι πως ο αριθμός συγκρούσεων έγινε 0, τότε ο αλγόριθμος επιστρέφει την δομή Arhikos_pinakas σαν λύση του προβλήματος, εκτυπώνει τα αποτελέσματα και τερματίζει.

Εάν ο λόγος εξόδου από το βρόγχο είναι η ολοκλήρωση του μέγιστου αριθμού επαναλήψεων, ο αλγόριθμος απέτυχε να βρει λύση. Ελέγχει αν έχει ολοκληρωθεί ο μέγιστος αριθμός επανεκκινήσεων. Αν ναι, τότε επιστρέφει αποτυχία και τερματίζει. Αν όχι, εκτελείται από την αρχή μέχρι να βρει λύση ή να ολοκληρωθούν οι μέγιστες δυνατές επανεκκινήσεις.

4.2.4. Σύνοψη

Ο αλγόριθμος Random-restarts είναι στην ουσία ένας αλγόριθμος min-conflicts που κάνει επανεκκινήσεις στο πρόβλημα .Με αυτό τον τρόπο ,γίνεται πιο δύσκολο να «κολλήσει» ο αλγόριθμος σε τοπικά μέγιστα, γιατί όταν κολλήσει στο πρώτο τοπικό μέγιστο θα γίνει επανεκκίνηση. Βέβαια, ακόμα υπάρχει η περίπτωση του τοπικού μέγιστου στην περίπτωση που τελειώσουν τα βήματα επανεκκίνησης. Το μειονέκτημα είναι ότι η μεταβλητή, το μάθημα δηλαδή, επιλέγεται τυχαία από το σύνολο των μεταβλητών που εμπλέκονται σε παραβίαση.

Κεφάλαιο 5

Πειραματική μελέτη

5.1 Εισαγωγή

Η αξιολόγηση του αλγόριθμου τοπικής αναζήτησης min-conflicts με random restarts είναι ο βασικός σκοπός της μελέτης αυτής. Στο κεφάλαιο αυτό θα εξεταστούν σε βάθος τα αποτελέσματα εκτέλεσης του αλγορίθμου αυτού για ένα δοσμένο πρόβλημα. Από τους πίνακες που θα ακολουθήσουν θα βγάλουμε κάποια συμπεράσματα για την απόδοσή του, τα οποία θα χρησιμέψουν σε περαιτέρω μελέτες. Επιπλέον, στόχος της μελέτης ήταν να διαπιστώσουμε κατά πόσο η μοντελοποίηση κι επίλυση του προβλήματος με μεθόδους ικανοποίησης περιορισμών είναι ικανή να προσφέρει αποδεκτές λύσεις σε ρεαλιστικά προβλήματα κατασκευής ωρολογίου προγράμματος.

Η εκτέλεση του αλγορίθμου έγινε για τριών ειδών προβλήματα: εύκολα, μέτρια και δύσκολα προβλήματα. Ο διαχωρισμός αυτός έγινε με βάση τον αριθμό των μαθημάτων προς εξέταση. Όσο λιγότερα είναι τα μαθήματα που πρέπει να μπουν σε χρονοσχισμές, τόσο πιο εύκολο θεωρείται το πρόβλημα. Η ίδια διαδικασία έγινε και για τις τρεις περιπτώσεις και τα αποτελέσματα κάθε περιπτώσεις διαφέρουν σημαντικά μεταξύ τους.

Η πειραματική διαδικασία έγινε σε στάδια. Κάθε φορά κρατάμε σταθερή μια τιμή του προβλήματος και αλλάζουμε τις υπόλοιπες, παρατηρώντας την απόδοση του αλγορίθμου. Για παράδειγμα, κρατάμε σταθερό τον μέγιστο επιτρεπτό αριθμό επαναλήψεων και μεταβάλλουμε τον αριθμό επιτρεπτών επανεκκινήσεων. Σε κάθε στάδιο, τρέχουμε το πρόγραμμα 50 φορές, ώστε να βγάλουμε ένα καλό μέσο όρο για τα αποτελέσματά μας.

Κατά την εκτέλεση του αλγορίθμου, παρατηρήθηκε ότι εμφανίζονταν διαφορετικά αποτελέσματα κάθε φορά που αλλάζαμε τα στοιχεία του προβλήματος. Ως στοιχεία εννοούμε, πχ τόσο τον μέγιστο αριθμό επανεκκινήσεων, όσο και πχ τον αριθμό των διαθέσιμων αιθουσών εξέτασης. Έτσι, για να βρεθούν αρχικά οι παράμετροι του προβλήματος για τις οποίες η απόδοση του αλγορίθμου βελτιστοποιείται, χρησιμοποιήσαμε κάποια κριτήρια απόδοσης. Στη συνέχεια, μετά την εύρεση των βέλτιστων παραμέτρων, θα πραγματοποιήσουμε αλλαγές στο πρόβλημα για να παρατηρήσουμε την συμπεριφορά του αλγορίθμου.

5.2 Κριτήρια απόδοσης

Τα κριτήρια που επιλέχθηκαν για να εξεταστεί η απόδοση του αλγορίθμου είναι τα παρακάτω τρία:

- Ποσοστό επιτυχίας.
- Μέσος χρόνος εκτέλεσης
- Μέσο βήμα εκτέλεσης

Όπως μπορεί εύκολα να γίνει αντιληπτό, το ποσοστό επιτυχίας είναι το ποσοστό εύρεσης λύσης του αλγορίθμου εκφρασμένο σε ποσοστό % και αποτελεί και το βασικό κριτήριο για την απόδοσή του. Θα επιδιώξουμε να μεγιστοποιήσουμε το ποσοστό αυτό, έτσι ώστε να μας εγγυάται ο αλγόριθμος ότι θα βρει λύση.

Ο μέσος χρόνος εκτέλεσης με δείχνει το μέσο χρόνο που θα κάνει το πρόγραμμα να εκτελεστεί από τη στιγμή που ξεκινά ο αλγόριθμος μέχρι να τελειώσει. Στο χρόνο αυτό δεν περιλαμβάνονται μόνο τα βήματα της διαδικασίας εύρεσης λύσης, αλλά και τα επιπρόσθετα μέρη του προβλήματος, όπως η εμφάνιση των αρχικών στοιχείων και των τελικών αποτελεσμάτων. Τα μέρη αυτά αποτελούν μικρό μέρος του κώδικα, και ο χρόνος εκτέλεσής τους είναι ο ίδιος κάθε φορά. Έτσι το γεγονός ότι συμπεριλαμβάνονται στο συνολικό χρόνο εκτέλεσης δεν επηρεάζει τη σύγκριση των χρόνων. Οι τιμές του μέσου χρόνου εκτέλεσης μπορεί να ποικίλλουν από μερικά ms έως κάποια δευτερόλεπτα. Το κριτήριο του χρόνου μπορεί να μοιάζει περιττό στη περίπτωσή μας αφού μια μικρή διαφορά της τάξεως κάποιων ms ή κάποιων δευτερολέπτων δε φαίνεται να παίζει σημαντικό ρόλο. Παρόλα αυτά, ακόμα και μία τόσο μικρή διαφορά θα διογκωθεί και θα παίξει το ρόλο της σε περίπτωση που μεγαλώσει το πρόβλημα (για παράδειγμα να έχουμε 500 μαθήματα).

Με τον όρο μέσο βήμα εκτέλεσης εννοούμε τις επαναλήψεις και τις επανεκκινήσεις που θα χρειαστεί να κάνει ο αλγόριθμος μέχρι να βρει λύση. Όταν λέμε πχ ότι ο αλγόριθμος έτρεξε με μέσο βήμα εκτέλεσης 5,6-3,2, εννοούμε ότι κατά μέσο όρο χρειάστηκε 5,6 επαναλήψεις και 3,2 επανεκκινήσεις για να βρει λύση. Στο μέσο βήμα εκτέλεσης δεν περιλαμβάνονται οι εκτελέσεις στις οποίες ο αλγόριθμος απέτυχε να βρει λύση. Αναφέρεται μόνο στον μέσο αριθμό επαναλήψεων και επανεκκινήσεων που χρειάστηκαν στις περιπτώσεις που βρέθηκε λύση. Το κριτήριο αυτό δεν σχετίζεται καθόλου με το προηγούμενο, αφού όπως θα δούμε, μικρότερος μέσος χρόνος εκτέλεσης δεν εγγυάται ότι θα εκτελεστούν λιγότερες επαναλήψεις ή λιγότερες επανεκκινήσεις.

5.3 Πειραματικά αποτελέσματα

5.3.1 Εύκολο πρόβλημα

Στο πρώτο πρόβλημα έχουμε μόνο τέσσερα μαθήματα ανά εξάμηνο, γι' αυτό και θεωρείται εύκολο. Τα δεδομένα εισόδου φαίνονται στον παρακάτω πίνακα:

Όνομα μαθήματος	Επίθετο καθηγητή	Κωδικός καθηγητή	Εξάμηνο
Εισαγωγή στη πληροφορική	Αγγελίδης	0	1
Φυσική 1	Μακρίδης	1	1
Μαθηματική ανάλυση 1	Ζυγκιρίδης	2	1
Γραμμική άλγεβρα	Μπαλασάς	3	1
Αρχιτεκτονική Η/Υ	Δασυγένης	7	2
Πιθανότητες και στατιστική	Τσακιρίδου	8	2
Δίκτυα 1	Λάγκας	9	2
Εφαρμοσμένα μαθηματικά	Ζυγκιρίδης	2	2
Βάσεις δεδομένων	Συμεονίδης	14	3
Ψηφιακή επεξεργασία σημάτων	Κίτσας	15	3
Συστήματα επικοινωνιών 1	Παπαδόπουλος	16	3
Ανάλυση και σχεδίαση αλγορίθμων	Γεωργιάδης Λ	13	3
Συστήματα παράλληλης επεξεργασίας	Δασυγένης	7	4
Αναγνώριση προτύπων	Τζώρτζης	17	4
Αριθμητική ανάλυση	Δημητρόπουλος	18	4
Δίκτυα ευρείας ζώνης	Σαρηγιαννίδης	19	4
Προηγμένα θέματα λειτουργικών συστημάτων	Κοντάκη	24	5
Δίκτυα επικοινωνίας	Πετρίδου	25	5
Συστήματα κινητής υπολογιστικής	Δημόκας	26	5
Τεχνητή όραση	Δημητρόπουλος	18	5

Επειδή είναι μαθηματικά αδύνατο, αλλάξαμε τον περιορισμό που μας εξασφάλιζε ότι θα εξεταστούν τουλάχιστον δύο μαθήματα κάθε μέρα, καθώς ο αριθμός των μαθημάτων δεν επαρκεί (για τρεις εβδομάδες εξέτασης θα θέλαμε τουλάχιστον 30 μαθήματα για την οριακή περίπτωση). Ο περιορισμός μεταβλήθηκε ώστε να εξετάζεται τουλάχιστον ένα μάθημα κάθε μέρα και να μην υπάρχουν μέρες «κενές».

Αρχικά επιλέξαμε να διατηρήσουμε σταθερό τον μέγιστο αριθμό επαναλήψεων στη τιμή 20 και να αλλάξουμε την τιμή του αριθμού των επιτρεπτών επανεκκινήσεων μέχρι να βρούμε που μεγιστοποιείται η απόδοση. Τα αποτελέσματα των προσομοιώσεων φαίνονται στον παρακάτω πίνακα:

Επανεκκινήσεις	Ποσοστό επιτυχίας	Μέσος χρόνος εκτέλεσης	Μέσο βήμα εκτέλεσης
3	100%	0,245 s	2.40-0
5	100%	0,234 s	2.34-0

Από τον παραπάνω πίνακα διαπιστώνουμε ότι 100 επαναλήψεις είναι υπεραρκετές ώστε να μπορεί να βρει λύση ο αλγόριθμος χωρίς να χρειαστεί να κάνει επανεκκίνηση. Βρίσκει λύση στο 100% των περιπτώσεων σε μέσο χρόνο 0,245 δευτερολέπτων στη μία περίπτωση και 0,234 στη δεύτερη. Η διαφορά αυτή θεωρείται αμελητέα. Από το μέσο βήμα εκτέλεσης βλέπουμε ότι για να βρει λύση ο αλγόριθμος χρειάζεται κατά μέσο όρο 2,4 και 2,34 επαναλήψεις αντίστοιχα και καμία επανεκκίνηση.

Ο μόνος τρόπος να δούμε διαφορά σε αυτά τα αποτελέσματα είναι να μικρύνουμε πολύ τον αριθμό των επαναλήψεων και να τον φέρουμε κοντά στη μέση τιμή των βημάτων. Για να δειχθεί η αλλαγή, εκτελέσαμε το πείραμα για 5 επιτρεπτές επανεκκινήσεις και μέγιστο αριθμό επαναλήψεων 5. Τα αποτελέσματα φαίνονται παρακάτω:

Επανεκκινήσεις	Επαναλήψεις	Ποσοστό επιτυχίας	Μέσος χρόνος εκτέλεσης	Μέσο βήμα εκτέλεσης
5	5	92%	0,234 s	2.34-0,52

Παρόλο που το πρόβλημα είναι εύκολο στη συγκεκριμένη περίπτωση μπορούνε να κάνουμε κάποιες παρατηρήσεις:

- Ο αλγόριθμος Random-restarts βρίσκει πάντα λύση εκτός αν μικρύνουμε πολύ τον αριθμό των μέγιστων επαναλήψεων (5 για παράδειγμα). Σ αυτή τη περίπτωση βλέπουμε πως η απόδοση μειώνεται δραματικά, γεγονός όμως που μπορεί να αποφευχθεί εύκολα. Ακόμα και σε αυτή την οριακή περίπτωση, παρατηρούμε ότι κάνοντας κάποιες επανεκκινήσεις, ο αλγόριθμος πάλι καταφέρνει να βρει λύση στην πλειονότητα των περιπτώσεων.
- Ο μέσος χρόνος εύρεσης λύσης είναι πολύ μικρός εξαιτίας της απλότητας του προβλήματος και δεν επηρεάζεται από τον μέγιστο αριθμό των επανεκκινήσεων, αφού σπάνια ο αλγόριθμος δεν βρίσκει λύση με την πρώτη προσπάθεια.

5.3.2 Μέτριο πρόβλημα

Στη δεύτερη περίπτωση έχουμε φυσιολογικό αριθμό μαθημάτων ανά εξάμηνο, που κυμαίνεται στα 6-7, γι’ αυτό και θεωρείται μέτριο. Τα δεδομένα εισόδου φαίνονται στον παρακάτω πίνακα:

Όνομα μαθήματος	Επίθετο καθηγητή	Κωδικός καθηγητή	Εξάμηνο
Εισαγωγή στη πληροφορική	Αγγελίδης	0	1
Φυσική 1	Μακρίδης	1	1
Μαθηματική ανάλυση 1	Ζυγκιρίδης	2	1
Γραμμική άλγεβρα	Μπαλασάς	3	1
Αγγλικά 1	Γαλάνη	4	1
Εισαγωγή στο δομημένο προγραμματισμό	Στεργίου	5	1
Εισαγωγή στις τηλεπικοινωνίες	Παπαδημητρίου	6	1
Αρχιτεκτονική Η/Υ	Δασυγένης	7	2
Πιθανότητες και στατιστική	Τσακιρίδου	8	2
Δίκτυα 1	Λάγκας	9	2
Εφαρμοσμένα μαθηματικά	Ζυγκιρίδης	2	2
Αρχές οικονομικής επιστήμης	Μπακούρος	11	2
Ψηφιακή σχεδίαση 1	Δημητρακόπουλος	12	2
Βάσεις δεδομένων	Συμεονίδης	14	3
Ψηφιακή επεξεργασία σημάτων	Κίτσας	15	3
Συστήματα επικοινωνιών 1	Παπαδόπουλος	16	3
Ανάλυση και σχεδίαση αλγορίθμων	Γεωργιάδης Λ	13	3
Ηλεκτρονική 2	Δημητρακόπουλος	12	3
Ηλεκτρομαγνητικά πεδία	Μακρίδης	1	3
Συστήματα παράλληλης επεξεργασίας	Δασυγένης	7	4
Αναγνώριση προτύπων	Τζώρτζης	17	4
Αριθμητική ανάλυση	Δημητρόπουλος	18	4
Δίκτυα ευρείας ζώνης	Σαρηγιαννίδης	19	4
Ηλεκτρονική υγεία	Αγγελίδης	0	4
Τεχνητή νοημοσύνη	Στεργίου	5	4
Μικροκυματικές επικοινωνίες	Βεργάδος	20	4
Προηγμένα θέματα λειτουργικών συστημάτων	Κοντάκη	24	5
Δίκτυα επικοινωνίας	Πετρίδου	25	5
Συστήματα κινητής υπολογιστικής	Δημόκας	26	5
Τεχνητή όραση	Δημητρόπουλος	18	5
Θεωρία πολυπλοκότητας	Τσαλικάκης	27	5
Νευρωνικά δίκτυα	Παρτάλας	28	5
Ηλεκτρονική επιχειρηματικότητα	Μπακούρος	11	5

Αρχικά επιλέξαμε να διατηρήσουμε σταθερό τον μέγιστο αριθμό επανεκκινήσεων στη τιμή 20 και να αλλάξουμε την τιμή του αριθμού των επιτρεπτών επαναλήψεων μέχρι να βρούμε που μεγιστοποιείται η απόδοση. Τα αποτελέσματα των προσομοιώσεων φαίνονται στον παρακάτω πίνακα:

Επαναλήψεις	Ποσοστό επιτυχίας	Μέσος χρόνος εκτέλεσης	Μέσο βήμα εκτέλεσης
20	36%	1,071 s	12,06-7,06
40	78%	1,772 s	15,15-6,64
60	92%	1,676 s	25,43-3,93
80	98%	1,990 s	35,34-3,83

Από τον παραπάνω πίνακα μπορούμε να κάνουμε τις εξής παρατηρήσεις:

- Ακόμα και σε ένα μέτριο πρόβλημα, ο Random-restarts καταφέρνει να βρει λύση σε μεγάλο ποσοστό, αρκεί να έχει στη διάθεσή του ένα ικανοποιητικό αριθμό επιτρεπτών επαναλήψεων. Όπως βλέπουμε, το ποσοστό ξεπερνά το 90% σε δύο περιπτώσεις παρόλο που έχουμε μόνο 20 διαθέσιμες επανεκκινήσεις.
- Ο χρόνος εύρεσης λύσης παραμένει μικρός, στις τάξεις του ενός με δύο δευτερολέπτων παρόλο που αυξήθηκε η δυσκολία του προβλήματος.
- Το ποσοστό αυξάνεται σημαντικά ακόμα και με μικρή αύξηση του αριθμού των επιτρεπτών επαναλήψεων. Αυτό μας δείχνει ότι στον Random-restarts κάθε επανάληψη παίζει σημαντικό ρόλο, αφού λίγες επαναλήψεις επιπλέον μπορεί να καθορίσουν αν θα βρεθεί λύση ή όχι.
- Καθώς αυξάνεται ο αριθμός των επαναλήψεων, βλέπουμε στο μέσο βήμα εκτέλεσης, ότι ο μέσος όρος επαναλήψεων που απαιτούνται για να βρεθεί λύση αυξάνεται, πράγμα που εξηγείται από το γεγονός ότι ο αλγόριθμος έχει στη διάθεσή του περισσότερες επαναλήψεις. Έτσι δεν καταλήγει γρήγορα σε επανεκκίνηση, αλλά βρίσκει πιο συχνά λύση με περισσότερες επαναλήψεις. Ταυτόχρονα, βλέπουμε ότι για τον ίδιο λόγο μειώνεται σταδιακά ο μέσος όρος επανεκκινήσεων που απαιτούνται για την εύρεση λύσης. Από τη στιγμή που ο αλγόριθμος βρίσκει πιο συχνά λύση με περισσότερες επαναλήψεις, αυτό συνεπάγεται ότι απαιτούνται λιγότερες επανεκκινήσεις.

Στο επόμενο βήμα της πειραματικής μας μελέτης, διατηρούμε σταθερή τη τιμή των επιτρεπτών επαναλήψεων στις 80 μεταβάλλοντας τον μέγιστο αριθμό επιτρεπτών επανεκκινήσεων και παρατηρούμε τη συμπεριφορά του αλγορίθμου.

Τα αποτελέσματα των πειραμάτων φαίνονται παρακάτω:

Επανεκκινήσεις	Ποσοστό επιτυχίας	Μέσος χρόνος εκτέλεσης	Μέσο βήμα εκτέλεσης
10	92%	2,245 s	35,73-2,73
20	98%	1,990 s	35,34-2,83
30	100%	1,676 s	34,98-2,96
40	100%	1,636 s	35,88-2,32

Από τις παραπάνω μετρήσεις, οι παρατηρήσεις που μπορούμε να κάνουμε είναι οι εξής:

- Αυξάνοντας τον αριθμό των επανεκκινήσεων βλέπουμε ότι βελτιώνεται σταδιακά το ποσοστό επιτυχίας του αλγορίθμου έως ότου σε κάποιο σημείο, οι διαθέσιμες επανεκκινήσεις επαρκούν για να εγγυηθούν ότι θα βρίσκεται πάντα λύση.

- Ο μέσος χρόνος εκτέλεσης παραμένει μικρός, στις τάξεις του ενός με δύο δευτερολέπτων. Ταυτόχρονα, βλέπουμε ότι καθώς αυξάνεται το ποσοστό επιτυχίας, μειώνεται ο μέσος χρόνος εκτέλεσης. Ο λόγος για τη συγκεκριμένη μείωση είναι ότι όταν το πρόγραμμα βρίσκει λύση περισσότερες φορές, τερματίζει πιο γρήγορα από το να κάνει όλες τις διαθέσιμες επαναλήψεις και επανεκκινήσεις. Έτσι, εφόσον βρίσκει περισσότερες φορές λύση τερματίζει περισσότερες φορές πιο νωρίς, με αποτέλεσμα να μειώνεται ο μέσος όρος του χρόνου εκτέλεσης.
- Τέλος βλέπουμε ότι το μέσο βήμα εκτέλεσης δεν επηρεάζεται από την αλλαγή του αριθμού των επανεκκινήσεων. Κάθε φορά που ο αλγόριθμος βρίσκει λύση χρειάζεται κατά μέσο όρο περίπου 30-35 επαναλήψεις και 2-3 επανεκκινήσεις, ανεξάρτητα από τον συνολικό αριθμό των επανεκκινήσεων που έχει στη διάθεσή του.

5.3.3 Δύσκολο πρόβλημα

Τέλος, θα εξετάσουμε την περίπτωση που έχουμε μεγαλύτερο αριθμό μαθημάτων ανά εξάμηνο σε σχέση με τα πραγματικά στοιχεία, για να δούμε πως συμπεριφέρεται ο αλγόριθμος όταν του δοθεί ένα δύσκολο πρόβλημα. Για το λόγο αυτό, θα εισάγουμε δύο με τρία υποθετικά νέα μαθήματα σε κάθε εξάμηνο, ώστε να έχουμε εννιά μαθήματα σε κάθε εξάμηνο. Τα δεδομένα εισόδου θα πάρουν τη παρακάτω μορφή:

Όνομα μαθήματος	Επίθετο καθηγητή	Κωδικός καθηγητή	Εξάμηνο
Εισαγωγή στη πληροφορική	Αγγελίδης	0	1
Φυσική 1	Μακρίδης	1	1
Μαθηματική ανάλυση 1	Ζυγκιρίδης	2	1
Γραμμική άλγεβρα	Μπαλασάς	3	1
Αγγλικά 1	Γαλάνη	4	1
Εισαγωγή στο δομημένο προγραμματισμό	Στεργίου	5	1
Εισαγωγή στις τηλεπικοινωνίες	Παπαδημητρίου	6	1
Μαθ1	Καθ1	31	1
Μαθ2	Καθ2	32	1
Αρχιτεκτονική Η/Υ	Δασυγένης	7	2
Πιθανότητες και στατιστική	Τσακιρίδου	8	2
Δίκτυα 1	Λάγκας	9	2
Εφαρμοσμένα μαθηματικά	Ζυγκιρίδης	2	2
Αρχές οικονομικής επιστήμης	Μπακούρος	11	2
Ψηφιακή σχεδίαση 1	Δημητρακόπουλος	12	2
Μαθ3	Καθ3	33	2
Μαθ4	Καθ4	34	2
Μαθ5	Καθ5	35	2
Βάσεις δεδομένων	Συμεονίδης	14	3
Ψηφιακή επεξεργασία σημάτων	Κίτσας	15	3
Συστήματα επικοινωνιών 1	Παπαδόπουλος	16	3

Ανάλυση και σχεδίαση αλγορίθμων	Γεωργιάδης Λ	13	3
Ηλεκτρονική 2	Δημητρακόπουλος	12	3
Ηλεκτρομαγνητικά πεδία	Μακρίδης	1	3
Μαθ6	Καθ6	36	3
Μαθ7	Καθ7	37	3
Μαθ8	Καθ8	38	3
Συστήματα παράλληλης επεξεργασίας	Δασυγένης	7	4
Αναγνώριση προτύπων	Τζώρτζης	17	4
Αριθμητική ανάλυση	Δημητρόπουλος	18	4
Δίκτυα ευρείας ζώνης	Σαρηγιαννίδης	19	4
Ηλεκτρονική υγεία	Αγγελίδης	0	4
Τεχνητή νοημοσύνη	Στεργίου	5	4
Μικροκυματικές επικοινωνίες	Βεργάδος	20	4
Μαθ9	Καθ9	39	4
Μαθ10	Καθ10	40	4
Προηγμένα θέματα λειτουργικών συστημάτων	Κοντάκη	24	5
Δίκτυα επικοινωνίας	Πετρίδου	25	5
Συστήματα κινητής υπολογιστικής	Δημόκας	26	5
Τεχνητή όραση	Δημητρόπουλος	18	5
Θεωρία πολυπλοκότητας	Τσαλικάκης	27	5
Νευρωνικά δίκτυα	Παρτάλας	28	5
Ηλεκτρονική επιχειρηματικότητα	Μπακούρος	11	5
Μαθ11	Καθ11	41	5
Μαθ12	Καθ12	42	5

Αρχικά διατηρούμε σταθερό τον μέγιστο αριθμό επανεκκινήσεων στη τιμή 20 και αλλάζουμε την τιμή του αριθμού των επιτρεπτών επαναλήψεων μέχρι να βρούμε που μεγιστοποιείται η απόδοση. Τα αποτελέσματα των προσομοιώσεων φαίνονται στον παρακάτω πίνακα:

Επαναλήψεις	Ποσοστό επιτυχίας	Μέσος χρόνος εκτέλεσης	Μέσο βήμα εκτέλεσης
20	66%	2,330 s	16,39-7,36
40	86%	1,953 s	19,27-3,95
60	90%	1,780 s	26,04-2,49
80	96%	1,736 s	41,54-2,46

Τα αποτελέσματα που εξάγονται από τον παραπάνω πίνακα είναι τα εξής:

- Ακόμα και με την αύξηση της δυσκολίας του προβλήματος, βλέπουμε ότι ο αλγόριθμός μας μπορεί και βρίσκει λύση σε μεγάλο ποσοστό, αρκεί να έχει στη διάθεσή του ικανοποιητικό αριθμό επιτρεπτών επαναλήψεων. Όπως είδαμε και στο μέτριο πρόβλημα, έτσι και εδώ, το ποσοστό επιτυχίας

ξεπερνάει το 90% σε δύο περιπτώσεις, με την αύξηση του αριθμού των επαναλήψεων να είναι είκοσι σε κάθε βήμα. Αν συγκρίνουμε τα ποσοστά αυτά με τα ποσοστά του μέτριου προβλήματος, βλέπουμε πως παρόλο που η δυσκολία αυξήθηκε, τα ποσοστά επιτυχίας στις περιπτώσεις που δεν υπάρχει μεγάλος αριθμός επαναλήψεων, αυξήθηκαν επίσης. Το γεγονός αυτό συμβαίνει εξαιτίας του περιορισμού που επιβάλει να εξετάζονται τουλάχιστον δύο μαθήματα κάθε μέρα. Στο μέτριο πρόβλημα, ο μικρότερος αριθμός των μαθημάτων σε συνδυασμό με τις λίγες επαναλήψεις οδηγούσε τον αλγόριθμο σε οριακές καταστάσεις η και σε αποτυχίες πιο συχνά. Με την προσθήκη περισσότερων μαθημάτων ο περιορισμός αυτός ξεπεράστηκε εύκολα και τα ποσοστά επιτυχίας αυξήθηκαν. Στις περιπτώσεις, όμως, με αρκετό αριθμό επαναλήψεων, όπου ο αλγόριθμος μπορούσε να αντιμετωπίσει την παραβίαση του συγκεκριμένου περιορισμού χωρίς να ολοκληρώσει τον μέγιστο αριθμό επιτρεπτών επαναλήψεων, βλέπουμε ότι τα ποσοστά μειώθηκαν με την αύξηση της δυσκολίας, πράγμα φυσιολογικό.

- Ο χρόνος εύρεσης λύσης παραμένει για μια ακόμη φορά, πολύ μικρός και οι μεταβολές του από στάδιο σε στάδιο είναι αμελητέες. Παρατηρούμε, επίσης, ότι ο χρόνος μειώνεται όσο αυξάνεται το ποσοστό επιτυχίας, αν και όπως είπαμε η διαφορά είναι ανεπαίσθητη. Αναφορικά με το χρόνο, τέλος, μπορούμε να παρατηρήσουμε μια μικρή αύξηση σε σχέση με τους χρόνους εκτέλεσης του μέτριου προβλήματος, γεγονός που οφείλεται στην αύξηση των μεταβλητών του προβλήματος και επομένως στο χρόνο επεξεργασίας τους.
- Τέλος, όσο αφορά το μέσο βήμα εκτέλεσης, όπως εξηγήσαμε και στο μέτριο πρόβλημα, παρατηρούμε ότι όσο αυξάνουμε τις επιτρεπτές επαναλήψεις, τόσο αυξάνεται ο μέσος όρος επαναλήψεων και μειώνεται ο μέσος όρος επανεκκινήσεων μέχρι την εύρεση λύσης.

Στο επόμενο στάδιο της μελέτης, διατηρούμε σταθερή τη μέγιστη τιμή των επιτρεπτών επαναλήψεων στις 80 και μεταβάλλουμε κάθε φορά τον μέγιστο αριθμό επιτρεπτών επανεκκινήσεων, παρατηρώντας τη συμπεριφορά του αλγορίθμου.

Τα αποτελέσματα των πειραμάτων φαίνονται παρακάτω:

Επανεκκινήσεις	Ποσοστό επιτυχίας	Μέσος χρόνος εκτέλεσης	Μέσο βήμα εκτέλεσης
10	94%	1,795 s	39,04-2,83
20	96%	1,736 s	38,54-2,46
30	100%	1,683 s	37,04-2,64
40	100%	1,697 s	38,88-2,52

Παρατηρώντας τον παραπάνω πίνακα διαπιστώνουμε ότι:

- Αυξάνοντας τον αριθμό των επιτρεπτών επανεκκινήσεων, ο αλγόριθμός μας βελτιώνεται περεταίρω, το ποσοστό επιτυχίας του αυξάνεται σταδιακά, μέχρι τη στιγμή που φτάνουμε σε σημείο όπου θα βρίσκεται πάντα λύση στο πρόβλημά μας.
- Ο μέσος χρόνος εκτέλεσης μειώνεται όσο αυξάνεται το ποσοστό εύρεσης λύσης, αφού ο αλγόριθμος βρίσκει λύση περισσότερες φορές και δεν αναλώνεται σε περιττές επαναλήψεις και επανεκκινήσεις, αλλά η μείωση του χρόνου αυτού είναι ελάχιστη και δεν σοβαρό κριτήριο για τη σύγκριση των περιπτώσεων.

- Τέλος, βλέπουμε επίσης στο μέσο βήμα εκτέλεσης, ότι τόσο ο μέσος αριθμός επαναλήψεων όσο και ο μέσος αριθμός επανεκκινήσεων δεν μεταβάλλονται με την αύξηση του επιτρεπτού αριθμού επανεκκινήσεων. Παρόλα αυτά παρατηρούμε αύξηση του μέσου αριθμού επαναλήψεων σε σχέση με τα στοιχεία του μέτριου προβλήματος, γεγονός που εξηγείται από την αύξηση της δυσκολίας του προβλήματος.

5.3.4 Αλλαγή στοιχείων εξέτασης στο πρόβλημα

Αφού βρήκαμε τις τιμές των μεταβλητών επαναλήψεων και επανεκκινήσεων για τις οποίες βελτιστοποιείται η απόδοση του αλγόριθμου Random-restarts, ένας ακόμη τρόπος για να παρατηρήσουμε τη συμπεριφορά του είναι η αλλαγή των στοιχείων εξέτασης που κρατούσαμε σταθερά μέχρι τώρα. Τα στοιχεία αυτά αφορούν τις συνολικές μέρες που θα διαρκέσει η εξεταστική περίοδος, τις συνολικές διαθέσιμες ώρες εξέτασης κάθε μέρας και τις διαθέσιμες αίθουσες στις οποίες θα πραγματοποιηθεί η εξέταση.

Η μεθοδολογία των πειραμάτων που θα δούμε στη συνέχεια παραμένει ίδια με τη προηγούμενη. Σε κάθε στάδιο των πειραμάτων, θα διατηρούμε σταθερές τις μεταβλητές, στις τιμές για τις οποίες είδαμε ότι ο αλγόριθμος έχει τα βέλτιστα αποτελέσματα και θα αλλάζουμε κάθε φορά ένα από τα στοιχεία της εξέτασης, συλλέγοντας πληροφορίες και παρατηρώντας τη συμπεριφορά του αλγορίθμου. Οι τιμές που κρατήσαμε σταθερές είναι 40 επιτρεπτές επανεκκινήσεις και 80 επιτρεπτές επαναλήψεις. Τα παρακάτω πειράματα θα εφαρμοστούν στον κώδικα του μέτριου προβλήματος, καθώς αυτό αντιπροσωπεύει ένα πραγματικό πρόβλημα.

Στο πρώτο στάδιο των πειραμάτων μεταβάλαμε τη τιμή των διαθέσιμων ημερών εξέτασης. Για κάθε τιμή «τρέξαμε» το πρόγραμμα 50 φορές ώστε να βγάλουμε ένα ικανοποιητικό μέσο όρο για τα αποτελέσματά μας. Τα αποτελέσματα που πήραμε φαίνονται στον παρακάτω πίνακα:

Ημέρες	Ποσοστό επιτυχίας	Μέσος χρόνος εκτέλεσης	Μέσο βήμα εκτέλεσης
10	100%	1,911 s	39,04-6,40
15	100%	1,636 s	35,88-2,32
20	100%	0,486 s	6,15-0

Τα παρακάτω αποτελέσματα βγήκαν μετά από μια αναγκαστική αλλαγή σε έναν από τους περιορισμούς. Στη τρίτη περίπτωση, αλλάξαμε το περιορισμό που αναγκάζει το πρόγραμμα να τοποθετεί τουλάχιστον δύο μαθήματα σε κάθε μέρα εξέτασης. Η αλλαγή που κάναμε ήταν να μειώσουμε τον αριθμό των υποχρεωτικών μαθημάτων ανά μέρα σε ένα, αφού ο αριθμός των μαθημάτων (33) δεν επαρκούσε για την εξέταση δύο μαθημάτων καθημερινά (χρειάζονται τουλάχιστον 40 μαθήματα).

Παρατηρώντας τον παραπάνω πίνακα διαπιστώνουμε ότι το πρόγραμμα συνεχίζει να βρίσκει λύση με ποσοστό επιτυχίας 100% σε όλες τις περιπτώσεις. Τα μόνα που στοιχεία που αλλάζουν είναι ο μέσος χρόνος εκτέλεσης και το μέσο βήμα εκτέλεσης. Ο μέσος χρόνος εκτέλεσης μειώνεται δραματικά καθώς αυξάνουμε τις διαθέσιμες μέρες εξέτασης, από περίπου δύο δευτερόλεπτα σε μερικά δέκατα του δευτερολέπτου. Στο μέσο βήμα εκτέλεσης παρατηρούμε μια παρόμοια δραματική πτώση των τιμών. Από περίπου 39 επαναλήψεις και 9 επανεκκινήσεις, ο αλγόριθμος καταλήγει να βρίσκει λύση με μόλις περίπου 6 επαναλήψεις και καμία επανεκκίνηση

κατά μέσο όρο. η αιτία για τις παραπάνω μειώσεις των τιμών είναι πως οι περισσότεροι περιορισμοί σχετίζονται με τη μέρα εξέτασης των μαθημάτων. Έτσι, αυξάνοντας τις διαθέσιμες μέρες εξέτασης, τα μαθήματα παραβιάζουν λιγότερους περιορισμού και έτσι αλγόριθμος, κάνει λιγότερες μετακινήσεις και τελειώνει σε λιγότερο χρόνο.

Στο δεύτερο στάδιο των πειραμάτων μεταβάλουμε την τιμή των διαθέσιμων ωρών εξέτασης. Όπως αναφέραμε και στο κομμάτι της μοντελοποίησης του προβλήματός μας, οι ώρες εξέτασης χωρίζονται σε τρίωρα. Επί τούτου, η αλλαγή που θα πραγματοποιήσουμε αφορά τον αριθμό των διαθέσιμων τρίωρων για εξέταση. Τα αποτελέσματα των πειραμάτων μας φαίνονται παρακάτω:

Τρίωρα	Ποσοστό επιτυχίας	Μέσος χρόνος εκτέλεσης	Μέσο βήμα εκτέλεσης
1	70%	1,153 s	15,68-8,77
2	92%	2,057 s	13,30-9,76
3	100%	2,564 s	14,36-8,98
4	100%	2,636 s	15,88-7,32
5	100%	3,004 s	13.09-6,04

Στον παραπάνω πίνακα μπορούμε να κάνουμε τις εξής παρατηρήσεις:

- Αν μειώσουμε πολύ το αριθμό των διαθέσιμων τρίωρων βλέπουμε πως το ποσοστό επιτυχίας του αλγορίθμου μας μειώνεται. Αντίθετα, αν αυξήσουμε τη τιμή από 3 και πάνω βλέπουμε πως θα βρεθεί λύση σε κάθε εκτέλεσή του.
- Παρατηρούμε μια αύξηση στο μέσο χρόνο εκτέλεσης, όσο αυξάνουμε τον αριθμό των τρίωρων. Η αιτία για αυτή την αύξηση είναι η αύξηση των χρονοσχισμών που πρέπει να επεξεργαστεί ο αλγόριθμος σε κάθε επανάληψη. Όπως εξηγήσαμε και παραπάνω, ο αλγόριθμος τοποθετεί σε κάθε επανάληψη ένα μάθημα σε όλες τις χρονοσχισμές. Αφού εμείς αυξάνουμε τον αριθμό των χρονοσχισμών, λογικά θα αυξηθεί και ο χρόνος που χρειάζεται ο αλγόριθμος για τις προσπελάσει όλες.
- Τέλος, στο μέσο βήμα εκτέλεσης, παρατηρούμε πως ο μέσος όρος επαναλήψεων και επανεκκινήσεων για την εύρεση λύσης δεν επηρεάζεται σημαντικά από την αύξηση ή τη μείωση του αριθμού των τρίωρων.

Στο τελευταίο στάδιο της πειραματικής μας μελέτης θα εξετάσουμε τι συμβαίνει στο αλγόριθμο όταν μεταβάλλουμε την τιμή των διαθέσιμων αιθουσών για τις εξετάσεις. Τα αποτελέσματα που πήραμε από τα πειράματά μας συγκεντρώθηκαν στο ν παρακάτω πίνακα:

Αίθουσες	Ποσοστό επιτυχίας	Μέσος χρόνος εκτέλεσης	Μέσο βήμα εκτέλεσης
1	68%	1,180 s	17,32-11,44
2	86%	1,604 s	14,41-8,16
3	92%	1,972 s	14,58-9,04
4	100%	3,086 s	16,42-10,46
5	100%	2,636 s	15.88-7,32

Από τα στοιχεία του παραπάνω πίνακα μπορούμε να εξάγουμε τα εξής συμπεράσματα:

- Η μείωση του αριθμού των διαθέσιμων αιθουσών, βλέπουμε πως έχει αντίκτυπο στο ποσοστό επιτυχίας του αλγορίθμου. Εγγυημένα εύρεση λύση έχουμε μόνο για τουλάχιστον τέσσερις αίθουσες. Για λιγότερες, παρατηρούμε ότι το ποσοστό μειώνεται σημαντικά.
- Ο μέσος χρόνος εκτέλεσης αυξάνεται όσο αυξάνεται ο αριθμός των αιθουσών. Η αιτία είναι, όπως είπαμε και παραπάνω, η αύξηση του αριθμού των συνολικών χρονοθυρίδων, που συνεπάγεται περισσότερο χρόνο για την προσπέλασή τους.
- Τέλος, στο μέσο βήμα εκτέλεση δεν παρατηρείται καμία σημαντική μεταβολή ανάμεσα στα αποτελέσματα των πέντε σεναρίων. Σε κάθε στάδιο, ο αλγόριθμος χρειάζεται, κατά μέσο όρο, περίπου 14 έως 17 επαναλήψεις και 7 έως 11 επανεκκινήσεις για να βρει λύση.

Κεφάλαιο 6

Συμπεράσματα

Όπως είχε αναφερθεί και παραπάνω, στόχος της εργασίας αυτής αποτελεί η παραγωγή ενός ωρολογίου προγράμματος εξεταστικής περιόδου το οποίο να προσεγγίζει ένα αντίστοιχο ρεαλιστικό. Μέσα από την προσπάθεια αυτή υλοποιήθηκε ο αλγόριθμος Random Restarts με τυχαίες επανεκκινήσεις. Εκτελέστηκε για διάφορα σενάρια, σε καθένα από τα οποία μεταβαλλόταν κάποιο στοιχείο του προβλήματός μας, για να μελετηθεί η συμπεριφορά του και να βρεθούν οι τιμές των μεταβλητών για τις οποίες βελτιστοποιήθηκε η απόδοσή του. Συγκρίθηκαν τα αποτελέσματα κάθε σεναρίου ως προς το ποσοστό επιτυχίας εύρεσης λύσης, τον μέσο χρόνο εκτέλεσης του προγράμματος, καθώς και τον μέσο αριθμό των επαναλήψεων και επανεκκινήσεων που χρειάζονται μέχρι την εύρεσή της.

Το τελικό ωρολόγιο πρόγραμμα εξεταστικής που δημιουργήθηκε στα πλαίσια της εργασίας αυτής καλύπτει σε ικανοποιητικό βαθμό τις βασικές ανάγκες του τμήματος Μηχανικών Πληροφορικής και Τηλεπικοινωνιών. Γενικά, οι αλγόριθμοι τοπικής αναζήτησης έχουν πολύ καλή απόδοση σε προβλήματα που δεν έχουν μεγάλο μέγεθος, όπως της συγκεκριμένης σχολής. Σίγουρα όμως, υπάρχουν ακόμη δυνατότητες βελτίωσης του προγράμματος αυτού με διάφορους τρόπους. Κάποιοι από αυτούς είναι οι παρακάτω:

- Κατ' αρχήν, μπορούμε να θεωρήσουμε το παρόν πρόβλημα ως ένα πρόβλημα βελτιστοποίησης. Αυτό σημαίνει ότι μπορούμε να εισάγουμε «μαλακούς» περιορισμούς ή αλλιώς προτιμήσεις είτε των καθηγητών, είτε των φοιτητών, οι οποίες κατά την παραγωγή του προγράμματος θα λαμβάνονται υπόψη για τη παραγωγή του τελικού αποτελέσματος. Ένα παράδειγμα τέτοιου περιορισμού είναι η προτίμηση κάποιου καθηγητή για εξέταση του μαθήματός του μια συγκεκριμένη ημερομηνία. Ο συγκεκριμένος τρόπος συμβάλλει σημαντικά στην παραγωγή ενός ποιοτικότερου προγράμματος.
- Επίσης, ένας ακόμη τρόπος βελτίωσης του προβλήματος είναι η υλοποίησή του με διαφορετικούς αλγορίθμους, οι οποίοι μπορεί να παράγουν τα αποτελέσματα ακόμη πιο γρήγορα και για πιο πολύπλοκα παρόμοια προβλήματα.
- Επιπλέον, σε κάποια μελλοντική μελέτη θα μπορούσε να γίνει μια πιο αυστηρή μοντελοποίηση του προβλήματος. Στην παρούσα εργασία κάναμε κάποιες παραδοχές για να διευκολυνθούμε στο προγραμματιστικό κομμάτι. Τέτοιες παραδοχές ήταν, για παράδειγμα, να χωρίσουμε τις ώρες των εξετάσεων σε τρίωρα ή να μην θεωρήσουμε ότι υπάρχει περιορισμός χωρητικότητας στις αίθουσες. Παραδοχές σαν τις παραπάνω θα μπορούσαν να αναλυθούν περαιτέρω και να οδηγήσουν σε ένα ακόμα πιο αληθοφανές αποτέλεσμα.
- Τέλος, θα μπορούσε να γίνει το πρόγραμμα δυναμικό. Με τον όρο αυτό εννοούμε την ικανότητά του να αλλάζει και να βρίσκει άμεσα εναλλακτικές λύσεις σε περίπτωση που συμβεί κάτι απρόοπτο, όπως μια απουσία κάποιου καθηγητή ή μια βλάβη με μεγάλη διάρκεια που θα ακυρώσουν την εξέταση κάποιου μαθήματος. Αυτό είναι και το δυσκολότερο κομμάτι της υλοποίησης.

Βιβλιογραφία

1 "Implementations of Resource Constraints in ILOG SCHEDULE: A Library for the Development of Constraint –Based Scheduling Systems" Claude Le Pape, Intelligent Systems Engineering, 1994".
2 "Βελτιστοποίηση Χρονοπρογραμματισμού με Χρήση Γενετικών Αλγορίθμων, Αλέξανδρος Μ. Κελεμένης, 2003".
3 "Αλγόριθμοι Συνδυαστικής Βελτιστοποίησης με Έμφαση σε Μεταευρετικές Τεχνικές", ΧΡΗΣΤΟΣ Γ. ΓΚΟΓΚΟΣ, 2009.
4 "The Assignment Problem: A Math Programming Case Study".
5 "A Language for Specifying Complete Timetabling Problems", L.Reis, E.Oliveira".
6 A. Schaeft, "A Survey of Automated Timetabling", TR CS-R9567, CWI-Cent. Wiskunde en Informatica, 1995.
7 Carter M. and Laporte G., Recent Developments in Practical Examination Timetabling, 1996.Waltz, D. L., "Understanding line drawings of scenes with shadows, in: Psychology of Computer Vision", McGraw - Hill, New York, 1975.
8 Sutherland I., Sketchpad: a man-machine graphical communication system, in: Proc. IFIP Spring Joint Computer Conference, 1963.
9 Borning, A.: The Programming Language Aspects of Thing Lab, a Constraint-Oriented Simulation Laboratory, in: ACM Transactions on Programming Languages and Systems 3(4): 252-387, 1981.
10 "Constraint Programming: In Pursuit of the Holy Grail", Roman Bartak, 1998.
11 Waltz, D.L., "Understanding line drawings of scenes with shadows, in: Psychology of Computer Vision", McGraw-Hill, New York, 1975.
12 Kumar, V., "Algorithms for Constraint Satisfaction Problems: A Survey, AI Magazine" 13(1): 32-44, 1992.
13 Montanary, U.: Networks of constraints fundamental properties and applications to picture processing, in: Information Sciences 7: 95-132, 1974.
14 Haralick, R.M., Elliot, G.L.: Increasing tree search efficiency for constraint satisfaction problems, in: Artificial Intelligence 14:263-314, 1980.
15 Nilsson, N.J.: Principles of Artificial Intelligence, Tioga, Palo Alto, 1980.
16 Minton, S., Johnston, M.D., Laird, P.: Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems, in: Artificial Intelligence 58(1-3):161-206, 1992.
17 Wang, C, J., Tsang, E.P.K.: Solving constraint satisfaction problems using neural-networks, in: Proc. Second International Conference on Artificial Neural Networks, 1991.
18 Tsang, E.: Foundations of Constraint Satisfaction, Academic Press, London, 1995.
19 Lawler, E.W., Wood, D.E.: Branch-and-bound methods: a survey, in: Operations Research 14:699-719, 1966.
20 Freuder, E.C., Wallace, R.J.: Partial Constraint Satisfaction, in: Artificial Intelligence, 1-3(58): 21-70, 1992.
21 Borning, A., Duisberg, R., Freeman-Benson, B., Kramer, K., Wolf, M.: Constraint Hierarchies, in. Proc. ACM Conference on Object Oriented Programming Systems, Languages, and Applications, ACM, 1987.
22 Chan, P., Heus, K., Weil, G.: Nurse Scheduling with Global Constraints in CHIP: GYMNASTE, in: Proc of Practical Application of Constraint Technology (PACT98), London, UK, 1998.

23 Focacci, F., Lamma, E., Mello, P., Milano, M.: Constraint Logic Programming for the Crew Rostering Problem, in: Proc. of Practical Application of Constraint Technology (PACT97), London, UK, 1997.
24 ILOG, France, http://www.ilog.fr/.
25 Van Hentenryck, P., Michel, L., Deville, Y.: Numerica: A Modeling Language for Global Optimization, The MIT Press, Cambridge, Mass., 1997.
26 InSol Ltd., Israel, http://www.insol.co.il/.

Παράρτημα Α

Κώδικας C++

```
#include <iostream>
#include <iomanip>
#include <stdlib.h>
#include <fstream>
#include <ctime>
#include <time.h>
using namespace std;

class Plirofories {
	private:
	int ora;
	int mera;
	int aithousa;
	int sugrousi;
public:
	void set_mera(int m){mera = m;}
	int get_mera(){return mera;}
	void set_ora(int o){ora = o;}
	int get_ora(){return ora;}
	void set_aithousa(int a){aithousa = a;}
	int get_aithousa(){return aithousa;}
	void set_sugrousi(int s){sugrousi = s;}
	int get_sugrousi(){return sugrousi;}
};

class Stoixeia {
	private:
	std::string onoma_math;
	std::string onoma_kath;
	int kath;
	int eks;
public:
	void set_onoma_math(std::string onoma_m){onoma_math = onoma_m;}
	std::string get_onoma_math(){return onoma_math;}
	void set_onoma_kath(std::string onoma_k){onoma_kath = onoma_k;}
	std::string get_onoma_kath(){return onoma_kath;}
	void set_kath(int k){kath = k;}
	int get_kath(){return kath;}
	void set_eks(int e){eks = e;}
	int get_eks(){return eks;}
};
```

```
class Eksetastiki {
	private:
	Plirofories* Arhikos_pinakas;
	Stoixeia* Lista;
	int num;
public:
	void create_lista();
	void print_lista();
	void create_arhiko_pinaka();
	int eleghos_sugrouseon();
	void print_teliko_pinaka();
};


void Eksetastiki::create_lista(){
	std::string onoma_math;
	std::string onoma_kath;
	int kath;
	int eks;
	int count=0;
	ifstream eisodos1,eisodos2;
	eisodos1.open("test.txt");
	if(eisodos1 == NULL) {
		cout << "Apotuxia anoigmatos arxeiou. Eksodos" << endl;
	}
	while(!eisodos1.eof()) {
		eisodos1 >> onoma_math >> onoma_kath >> kath >> eks ;
		count++;
	}
	eisodos1.close();
	num = count;
	Lista = new Stoixeia[num];
	eisodos2.open("test.txt");
	if(eisodos2 == NULL) {
		cout << "Apotuxia anoigmatos arxeiou. Eksodos" << endl;
	}
	int i=0;
	while(!eisodos2.eof()) {
		eisodos2 >> onoma_math >> onoma_kath >> kath >> eks ;
		Lista[i].set_onoma_math(onoma_math);
		Lista[i].set_onoma_kath(onoma_kath);
		Lista[i].set_kath(kath);
		Lista[i].set_eks(eks);
		i++;
	}
	eisodos2.close();
}
```

```
void Eksetastiki::print_lista(){
	for(int i=0;i<num;i++) {
		cout << "To mathima: " << Lista[i].get_onoma_math() << " to kanei o
kathigitis: " << Lista[i].get_onoma_kath() << " sto eksamino: " << Lista[i].get_eks()+1 << endl;
	}
}

void Eksetastiki::create_arhiko_pinaka() {
	int temp;
	srand ((unsigned) time(0));
	Arhikos_pinakas = new Plirofories[num];
	for(int i=0;i<num;i++){
		temp = (rand()%15);
		Arhikos_pinakas[i].set_mera(temp);
		temp = (rand()%4);
		Arhikos_pinakas[i].set_ora(temp);
		temp = (rand()%4);
		Arhikos_pinakas[i].set_aithousa(temp);
		Arhikos_pinakas[i].set_sugrousi(0);
	}
}

int Eksetastiki::eleghos_sugrouseon() {
	int count_sugkrousewn=0;
	int count_mathimaton[15];
	int temp;
	for(int i=0;i<15;i++){
		count_mathimaton[i]=0;
	}
	int count_apostasewn[5];
	for(int i=0;i<5;i++){
		count_apostasewn[i]=0;
	}
	for (int i=0;i<num;i++) {
		count_mathimaton[Arhikos_pinakas[i].get_mera()]++;
		for (int j=i+1;j<num;j++) {
			if (Arhikos_pinakas[i].get_mera() == Arhikos_pinakas[j].get_mera()) {

				if (Lista[i].get_eks() == Lista[j].get_eks()) {
					count_sugkrousewn++;
					Arhikos_pinakas[i].set_sugrousi(1);
				}
				if (Arhikos_pinakas[i].get_ora() ==
Arhikos_pinakas[j].get_ora()) {
					if (Arhikos_pinakas[i].get_aithousa() ==
Arhikos_pinakas[j].get_aithousa()) {
						count_sugkrousewn++;
						Arhikos_pinakas[i].set_sugrousi(1);
					}
					else if (Lista[i].get_kath() == Lista[j].get_kath()) {
						count_sugkrousewn++;
```

```
                                        Arhikos_pinakas[i].set_sugrousi(1);
                                }
                        }
                }
                if (Lista[i].get_eks() == Lista[j].get_eks()) {
                        if ((Arhikos_pinakas[i].get_mera() -
Arhikos_pinakas[j].get_mera() == 1) || (Arhikos_pinakas[j].get_mera() -
Arhikos_pinakas[i].get_mera() == 1)) {
                                count_apostasewn[Lista[i].get_eks()]++;
                                if (count_apostasewn[Lista[i].get_eks()] > 3) {
                                        count_sugkrousewn++;
                                        Arhikos_pinakas[i].set_sugrousi(1);
                                }
                        }
                }
        }
    }
    for(int i=0;i<15;i++){
        if (count_mathimaton[i] < 2) {
            count_sugkrousewn++;
        }
        if (count_mathimaton[i] > 4) {
            for (int j=0;j<num;j++){
                if (Arhikos_pinakas[j].get_mera() == i){
                    Arhikos_pinakas[j].set_sugrousi(1);
                    count_sugkrousewn++;
                }
            }
        }
    }
    int max_epanalipseis=0;
    int epoxes1=0;
    int min_sugrouseis=count_sugkrousewn;
    int sugrouseis1;
    int temp_mera;
    int temp_ora;
    int temp_aithousa;
    int temp_sugrouseis[num];
    while ((max_epanalipseis<80) && (min_sugrouseis!=0)) {
        do {
            temp = (rand()%num);
            epoxes1++;
        }while ((Arhikos_pinakas[temp].get_sugrousi() != 1) && (epoxes1<1000));
        for (int i=0;i<num;i++) {
            Arhikos_pinakas[i].set_sugrousi(0);
        }
        count_mathimaton[Arhikos_pinakas[temp].get_mera()]--;
        for (int i=0;i<15;i++) {
            for (int j=0;j<4;j++) {
                for (int k=0;k<4;k++) {
                    Arhikos_pinakas[temp].set_mera(i);
```

```
                                        Arhikos_pinakas[temp].set_ora(j);
                                        Arhikos_pinakas[temp].set_aithousa(k);
                                        count_mathimaton[i]++;
                                        sugrouseis1=0;
                                        for(int g=0;g<5;g++){
                                                count_apostasewn[g]=0;
                                        }
                                        for(int p=0;p<15;p++){
                                                if (count_mathimaton[p] < 2) {
                                                        sugrouseis1++;
                                                }
                                                if (count_mathimaton[p] > 4) {
                                                        for (int m=0;m<num;m++){
                                                                if
(Arhikos_pinakas[m].get_mera() == p){

        Arhikos_pinakas[m].set_sugrousi(1);
                                                                        sugrouseis1++;
                                                                }
                                                        }

                                                }
                                        }
                                        for (int l=0;l<num;l++) {
                                        for (int n=l+1;n<num;n++) {
                                                        if (Arhikos_pinakas[l].get_mera() ==
Arhikos_pinakas[n].get_mera()) {
                                                          if (Lista[l].get_eks() ==
Lista[n].get_eks()) {
                                                                        sugrouseis1++;

        Arhikos_pinakas[l].set_sugrousi(1);
                                                                }
                                                                if
(Arhikos_pinakas[l].get_ora() == Arhikos_pinakas[n].get_ora()) {
                                                                        if
(Arhikos_pinakas[l].get_aithousa() == Arhikos_pinakas[n].get_aithousa()) {

        sugrouseis1++;

        Arhikos_pinakas[l].set_sugrousi(1);
                                                                        }
                                                                        else if
(Lista[l].get_kath() == Lista[n].get_kath()) {

        sugrouseis1++;

        Arhikos_pinakas[l].set_sugrousi(1);
                                                                        }
                                                                }
                                                        }
```

```
                                                    if (Lista[l].get_eks() ==
Lista[n].get_eks()) {
                                                        if
((Arhikos_pinakas[l].get_mera() - Arhikos_pinakas[n].get_mera() == 1) ||
(Arhikos_pinakas[n].get_mera() - Arhikos_pinakas[l].get_mera() == 1)) {

        count_apostasewn[Lista[l].get_eks()]++;
                                                            if
(count_apostasewn[Lista[l].get_eks()] > 3) {

        sugrouseis1++;

        Arhikos_pinakas[l].set_sugrousi(1);
                                                            }
                                                        }
                                                    }
                                                }
                                            }
                                            if (sugrouseis1 <= min_sugrouseis) {
                                                temp_mera=i;
                                                temp_ora=j;
                                                temp_aithousa=k;
                                                min_sugrouseis=sugrouseis1;
                                                for(int g=0;g<num;g++){
                                                    temp_sugrouseis[g] =
Arhikos_pinakas[g].get_sugrousi();
                                                }
                                            }
                                            count_mathimaton[i]--;
                                            for (int g=0;g<num;g++) {
                                                Arhikos_pinakas[g].set_sugrousi(0);
                                            }
                                        }
                                    }
                                }
                                Arhikos_pinakas[temp].set_mera(temp_mera);
                                Arhikos_pinakas[temp].set_ora(temp_ora);
                                Arhikos_pinakas[temp].set_aithousa(temp_aithousa);
                                count_mathimaton[temp_mera]++;
                                for(int g=0;g<num;g++){
                                    Arhikos_pinakas[g].set_sugrousi(temp_sugrouseis[g]);
                                }
                                max_epanalipseis++;
        }
        return min_sugrouseis;
}

void Eksetastiki::print_teliko_pinaka() { //ektupwsi telikou apotelesmatos

        for(int z=0;z<num;z++){
```

```
            cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " <<
Arhikos_pinakas[z].get_mera() << " tin ora " << Arhikos_pinakas[z].get_ora() << " stin
aithousa " << Arhikos_pinakas[z].get_aithousa() << endl;
        }
}

int main () {
        int max_epanekkiniseis=0;
        clock_t start = clock();
        int min_sygrouseis;
        Eksetastiki Ex;
        Ex.create_lista();
        Ex.print_lista();
        do {
                Ex.create_arhiko_pinaka();
                min_sygrouseis = Ex.eleghos_sugrouseon();
                max_epanekkiniseis++;
        } while (min_sygrouseis != 0 && max_epanekkiniseis < 40);
        cout << "epanekkiniseis: " << max_epanekkiniseis-1 << endl;
        if (min_sygrouseis ==0){
           Ex.print_teliko_pinaka();
           }
   else {
   cout << "Apotuxia" << endl;
   }
        clock_t ends = clock();
        cout << "Running Time : "<< (double)(ends - start) / CLOCKS_PER_SEC << endl;
        cout << "Press enter to exit";
        getchar();
        return 0;
}
```