ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΧΕΙΡΙΣΜΟΣ ΠΡΟΤΙΜΗΣΕΩΝ ΓΙΑ ΤΗΝ ΑΠΟΔΟΤΙΚΗ ΚΑΤΑΣΚΕΥΗ ΠΡΟΓΡΑΜΜΑΤΟΣ ΕΞΕΤΑΣΤΙΚΗΣ ΠΕΡΙΟΔΟΥ

ΧΡΗΣΤΟΣ ΜΠΑΪΡΑΜΟΓΛΟΥ

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ:
ΚΩΝΣΤΑΝΤΙΝΟΣ ΣΤΕΡΓΙΟΥ - *Επίκουρος Καθηγητής Π.Δ.Μ.*

Κοζάνη, Ιούλιος 2012

. . . . . .

Χρήστος Μπαϊράμογλου
Διπλωματούχος Μηχανικός Πληροφορικής και Τηλεπικοινωνιών Π.Δ.Μ.

Copyright © Χρήστος Μπαϊράμογλου, 2012.
Με επιφύλαξη παντός δικαιώματος.All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τους συγγραφείς.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τους συγγραφείς και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Πανεπιστημίου Δυτικής Μακεδονίας.

ΠΕΡΙΛΗΨΗ

Η παρούσα διπλωματική εργασία πραγματεύεται το πρόβλημα του χειρισμού προτιμήσεων για την αποδοτική κατασκευή ωρολογίου προγράμματος εξεταστικής περιόδου του πανεπιστημίου Δυτικής Μακεδονίας και συγκεκριμένα του τμήματος Μηχανικών Πληροφορικής και Τηλεπικοινωνιών. Λόγω της πολυπλοκότητας και της ύπαρξης πολυάριθμων μεταβλητών και παραμέτρων το πρόβλημα ανήκει στην κατηγορία των NP-complete προβλημάτων, γεγονός που καθιστά την εξεύρεση της βέλτιστης λύσης μια πολύ δύσκολη υπόθεση.

Στο **1ο Κεφάλαιο** περιγράφεται η έννοια του χρονοπρογραμματισμού, οι κατηγορίες του καθώς και οι δυσκολίες που προκύπτουν σε αυτές. Επίσης παρουσιάζονται οι μέθοδοι επίλυσης και το μαθηματικό μοντέλο του προβλήματος βελτιστοποίησης κατασκευής ωρολογίου προγράμματος εξεταστικής.

Στο **2ο Κεφάλαιο** αναφέρεται ο χρονοπρογραμματισμός ως πρόβλημα ικανοποίησης περιορισμών και αναλύονται ορισμένα τέτοια προβλήματα. Ακολουθεί η περιγραφή των ειδών των περιορισμών καθώς και η κατηγορία στην οποία ανήκει ο αλγόριθμος που θα χρησιμοποιηθεί για την επίλυση του προβλήματος.

Στο **3ο Κεφάλαιο** γίνεται η μοντελοποίηση του προγράμματος εξεταστικής ως CSP αναλύοντας τα δεδομένα και του περιορισμούς του προβλήματος και πως αυτά υλοποιούνται στον C++ κώδικα.

Στο **4ο Κεφάλαιο** επιλέγεται ο αλγόριθμος τοπικής αναζήτησης min-conflicts. Γίνεται μια περιγραφή του αλγορίθμου και πως αυτός συμβάλλει στην κωδικοποίηση του προγράμματος.

Στο **5ο Κεφάλαιο** παρουσιάζονται τρεις διαφορετικές τεχνικές χειρισμού των προτιμήσεων, ο περιορισμός κόστους, η συνάρτηση βελτιστοποίησης Fx και το Pareto Optimality, ώστε να βελτιωθεί το πρόγραμμα και να γίνει πιο ρεαλιστικό και πιο ποιοτικό, και γίνεται και σύγκριση τους μέσω πειραματικών αποτελεσμάτων.

Στο **6ο Κεφάλαιο** καταγράφονται τα συμπεράσματα από τη θεωρητική ανασκόπηση αλλά και την πρακτική αξία της διαδικασίας επίλυσης του προβλήματος.

Λέξεις Κλειδιά: χρονοπρογραμματισμός, χρονοπρογραμματισμός με περιορισμούς, CSP, Μοντελοποίηση Προβλήματος, min-conflicts, Περιορισμός Κόστους, Συνάρτηση Βελτιστοποίησης Fx, Pareto Optimality, Κώδικας C++

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να εκφράσω τις ειλικρινείς μου ευχαριστίες σε όλους όσους συνέβαλαν ώστε να έρθει εις πέρας η παρούσα προπτυχιακή διπλωματική εργασία.

Αισθάνομαι την ανάγκη να ευχαριστήσω θερμά τον Επίκουρο Καθηγητή και Επιβλέποντα της εργασίας αυτής κ. Κωνσταντίνο Στεργίου για την ευκαιρία που μου έδωσε να εργαστώ σε ένα σύγχρονο αντικείμενο και για τη συνεχή βοήθεια και καθοδήγησή του σε όλα τα στάδια διεκπεραίωσης της εργασίας.

Τέλος, ευχαριστώ την οικογένειά μου για την αγάπη, την συμπαράσταση, την κατανόηση και ανοχή της σε όλη τη διάρκεια της φοιτητικής μου διαδρομής.

Κοζάνη, Ιούλιος 2012

Χρήστος Μπαϊράμογλου

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΚΕΦΑΛΑΙΟ 1

ΧΡΟΝΟΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

1.1Εισαγωγή

Στην καθημερινότητά μας ερχόμαστε συνεχώς αντιμέτωποι με την ανάγκη να δημιουργήσουμε χρονοδιαγράμματα μέσα στα οποία θα εντάξουμε τις πολλαπλές δραστηριότητες μας που αφορούν στην εργασία μας, στις υποχρεώσεις του σπιτιού μας (π.χ. καθαριότητα, εξοπλισμός) και της οικογένειάς μας (π.χ. διάβασμα παιδιών, μεταφορά στο σχολείο), στις κοινωνικές υποχρεώσεις μας, ακόμα και στη διασκέδαση μας. Όλα τα παραπάνω για να διεξαχθούν αρμονικά πρέπει πρώτα να προγραμματιστούν και σίγουρα ένας σωστός προγραμματισμός συμβάλλει στην ποιότητα του αποτελέσματος άρα και στην ποιότητα της ζωής μας.

Κατά τον προγραμματισμό μας θα πρέπει να λαμβάνουμε υπ' όψιν μας όχι μόνο τους δικούς μας πόρους αλλά και των ανθρώπων με τους οποίους αλληλεπιδρούμε, τη διαθεσιμότητα των πόρων αυτών και τα πλαίσια μέσα στα οποία αυτοί μπορούν να χρησιμοποιηθούν. Έτσι θα καταφέρουμε να οδηγηθούμε στην καλύτερη επιλογή ταιριάσματος των διαθέσιμων πόρων και να εξασφαλίσουμε την ομαλή έκβαση των γεγονότων.

Ένα τέτοιο πρόβλημα αντιμετωπίζεται και στον εκπαιδευτικό χώρο με το χρονοπρογραμματισμό των διαλέξεων ή των εξετάσεων , που οφείλει να ικανοποιεί τόσο τους διδάσκοντες όσο και τους φοιτητές που συμμετέχουν στην παραγωγική αυτή πράξη της εκπαίδευσης και να εξασφαλίσει την αρμονική συνεργασία αυτών.

Έτσι για παράδειγμα, δε θα πρέπει να δημιουργούνται μεγάλα κενά ανάμεσα στις διαλέξεις ούτε πολλές ώρες συνεχόμενης παρακολούθησης, διότι απωθούν και δυσκολεύουν τους φοιτητές στη συμμετοχή τους στη διαδικασία. Επίσης στις εξετάσεις δε θα πρέπει να προγραμματίζονται πολλά εξεταζόμενα μαθήματα για την ίδια μέρα ή σε συνεχόμενες ώρες και ούτω καθεξής.

Ας ισχυριστούμε λοιπόν ότι μας ζητήθηκε να φτιάξουμε το πρόγραμμα εξεταστικής ενός πανεπιστημίου με διάρκεια τριών (3) εβδομάδων. Οι πρώτες μας ενέργειες θα ήταν να βρούμε το πλήθος των μαθημάτων στο οποίο υποχρεούνται οι φοιτητές να εξεταστούν και ποια είναι αυτά, οι καθηγητές που διδάσκουν τα

συγκεκριμένα μαθήματα, οι αίθουσες που είναι διαθέσιμες και κατάλληλες (π.χ. από την άποψη της χωρητικότητας) και πολλά ακόμη.

Στην προσπάθειά μας θα πρέπει να είμαστε πολύ προσεκτικοί να μην προγραμματίσουμε για παράδειγμα δύο μαθήματα του ίδιου εξαμήνου να εξετάζονται την ίδια ώρα, είτε να αναθέσουμε στον ίδιο διδάσκοντα δύο παραδόσεις την ίδια χρονική στιγμή ή να προγραμματίσουμε τη διεξαγωγή δύο εξετάσεων στην ίδια αίθουσα την ίδια ώρα. Θα πρέπει να εξασφαλίσουμε την χωρητικότητα των αιθουσών. Εκτός από αυτούς τους ισχυρούς περιορισμούς έχουμε να λάβουμε υπ' όψιν μας και αρκετούς ακόμα, τους χαλαρούς, που αφορούν για παράδειγμα στο ποια μέρα προτιμά ο καθηγητής να εξεταστεί το μάθημα που διδάσκει.

Από όλα τα παραπάνω συμπεραίνουμε ότι πρόκειται για ένα πολύπλοκο πρόβλημα το οποίο υπόκειται σε πολλούς περιορισμούς που οφείλουμε να ικανοποιήσουμε και η επίλυση του οποίου απαιτεί τον συνυπολογισμό πολλών παραγόντων και λεπτομερή σχεδιασμό.

1.2 Τι είναι χρονοπρογραμματισμός

Η έννοια του χρονοπρογραμματισμού περιγράφει την ανάγκη για το σχεδιασμό ενός ωρολογίου προγράμματος (***timetable***).Το 1996, ο Wren περιγράφει το πρόβλημα του χρονικού προγραμματισμού, ως το πρόβλημα της αξιοποίησης συγκεκριμένων πόρων, λαμβάνοντας υπ' όψιν την ύπαρξη κάποιων περιορισμών, σύμφωνα πάντα με περιορισμένο αριθμό χρονικών διαστημάτων, με σκοπό την ικανοποίηση αντικειμενικών στόχων στον υψηλότερο δυνατό βαθμό.

Τα προβλήματα του χρονικού προγραμματισμού εμφανίζουν υψηλή ποικιλομορφία και περικλείουν πολλούς και διαφορετικούς τομείς της ζωής μας. Ο ρόλος που διαδραματίζει ο ορθός σχεδιασμός του ωρολογίου προγράμματος (***timetable***) συχνά δεν γίνεται αντιληπτός, ή τουλάχιστον δεν είναι πάντα κατανοητός ο βαθμός στον οποίο ένα ωρολόγιο πρόγραμμα καθορίζει την καθημερινότητά μας.

Τον υψηλό βαθμό σημαντικότητας των ωρολογίων προγραμμάτων μπορούμε εύκολα να τον συμπεράνουμε, αν αναλογιστούμε την αξία τους και τη συμβολή στην ομαλή και αποδοτική λειτουργία και οργάνωση τομέων όπως η εκπαίδευση (ωρολόγια προγράμματα σχολείων – πανεπιστημίων), η υγεία (ωρολόγια

προγράμματα νοσηλευτών και χειρουργείων), οι μεταφορές (προγράμματα τρένων, λεωφορείων κ.α.), αλλά και ο αθλητισμός (προγράμματα αγώνων).

Στην πλειονότητά τους τα προβλήματα του χρονικού προγραμματισμού είναι πολύπλοκα, συνδυαστικά προβλήματα και μάλιστα στη γενική μορφή τους είναι NP Complete.Η πολυπλοκότητα αυτή καθιστά πολύ δύσκολη , έως και αδύνατη, την αποδοτική και βέλτιστη επίλυσή τους.

Το γεγονός αυτό ανάγει την επίλυση τέτοιων προβλημάτων σε μία πολύ δύσκολη, χρονοβόρα, επίπονη και συχνά ακριβή διαδικασία. Για το λόγω αυτό, επιχειρούνται διαφορετικές διατυπώσεις του προβλήματος, ώστε να γίνει εφικτή η επίλυσή του. Σε κάποιες περιπτώσεις, η λύση ενός προβλήματος χρονικού προγραμματισμού ουσιαστικά επικεντρώνεται, απλώς και μόνο στην εύρεση ενός ωρολογίου προγράμματος (***timetable***) που να ικανοποιεί όλους τους περιορισμούς. Σε αυτές τις περιπτώσεις το πρόβλημα διατυπώνεται ως **πρόβλημα αναζήτησης** (***search problem***). Σε κάποιες άλλες όμως περιπτώσεις το πρόβλημα μας διατυπώνεται ως **πρόβλημα βελτιστοποίησης** (***optimization problem***). Η απαίτηση που διατυπώνεται στο πρόβλημα βελτιστοποίησης είναι η εξεύρεση ενός ωρολογίου προγράμματος που να ικανοποιεί όλους τους «σκληρούς» περιορισμούς (***hard constraints***) και να ελαχιστοποιεί (ή να μεγιστοποιεί) μία αντικειμενική συνάρτηση που εμπεριέχει όλους τους «μαλακούς» περιορισμούς (***soft constraints***).

Και στις δύο περιπτώσεις (πρόβλημα αναζήτησης ή βελτιστοποίησης) ορίζουμε το θεμελιώδες πρόβλημα (***underlying problem***) που δεν είναι άλλο από το **πρόβλημα της επιλογής της λύσης.** Στα προβλήματα αναζήτησης η επιλογή γίνεται με βάση εάν υπάρχει μια λύση, ενώ στα προβλήματα βελτιστοποίησης το πρόβλημα της επιλογής έγκειται στο εάν υπάρχει μια λύση που να αποδίδει δεδομένη αξία-τιμή στην αντικειμενική συνάρτηση.

Ήδη έχουμε αναφερθεί στην πολυπλοκότητα των προβλημάτων του χρονικού προγραμματισμού. Στην ουσία όμως, αναφερόμαστε στην πολυπλοκότητα του θεμελιώδες προβλήματος της επιλογής. Το θεμελιώδες πρόβλημα ανήκει και αυτό στην κατηγορία των NP-Complete προβλημάτων σχεδόν για όλες τις παραλλαγές του προβλήματος. Για το λόγο αυτό, μια ακριβής λύση επιτυγχάνεται μόνο σε μικρού μεγέθους προβλήματα (π.χ. λιγότερα από δέκα (10) μαθήματα, για ένα ωρολόγιο πρόγραμμα πανεπιστημίου), αν και στην πραγματικότητα στα προβλήματα συνήθως εμπλέκονται μερικές εκατοντάδες μαθημάτων.

1.3 Κατηγορίες Προβλημάτων Χρονοπρογραμματισμού

Ένα πρόβλημα χρονοπρογραμματισμού (***timetabling problem***) αναφέρεται στον προγραμματισμό ενεργειών σε συγκεκριμένες χρονικές περιόδους και βάσει ορισμένων προϋποθέσεων, οι οποίες αφορούν τους πόρους που είναι διαθέσιμοι. Κλασσικό πρόβλημα χρονοπρογραμματισμού αποτελεί η εύρεση ενός ωρολογίου προγράμματος εξεταστικής μαθημάτων, που παραδίδουν καθηγητές σε ένα σύνολο μαθητών ή φοιτητών. Είναι ένα πρόβλημα που συναντάται αρκετά συχνά σε εκπαιδευτικά ιδρύματα, όπως σε σχολεία και πανεπιστήμια, καθώς τουλάχιστον μια φορά το χρόνο κατασκευάζεται ένα πρόγραμμα μαθημάτων σε ένα σχολείο και αρκετές σε ένα πανεπιστήμιο.

Σύμφωνα με τον A. Schaeft, μεγάλος αριθμός παραλλαγών του προβλήματος χρονοπρογραμματισμού έχει προταθεί, ανάλογα με τον τύπο του ιδρύματος που αφορά ένα πρόβλημα (πανεπιστήμιο ή σχολείο) και το είδος των περιορισμών που περιλαμβάνει Συγκεκριμένα διακρίνονται τρεις κατηγορίες προβλημάτων χρονοπρογραμματισμού: ο χρονοπρογραμματισμός σχολείου (***school timetabling***), ο χρονοπρογραμματισμός μαθημάτων πανεπιστημίου (***university course timetabling***) και ο χρονοπρογραμματισμός εξετάσεων (***examination timetabling***), οι οποίες θα περιγραφούν αναλυτικότερα ακολούθως.

Όπως παρατηρήθηκε από τον Schaeft,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, στη μελέτη του για τις πρόσφατες εξελίξεις στο πρακτικό timetabling εξετάσεων αναγνώρισε πέντε διαφορετικά υποπροβλήματα για το πρόβλημα timetabling μαθημάτων:

1. Timetabling μαθημάτων
2. Timetabling αίθουσας – καθηγητή
3. Προγραμματισμός μαθητών
4. Ανάθεση καθηγητών
5. 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 εξής:

1.Τα προβλήματα timetabling διαιρούνται ανάλογα με το βαθμό ελευθερίας στην τοποθέτηση παροχής πόρων και τα διαστήματα ζήτησης πόρων. Στα **καθαρά προβλήματα** η τοποθέτηση 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 η ίδια, ή και να ξεπερνά, τη διαθέσιμη χωρητικότητα

2.Διαφορετικά περιβάλλοντα έχουνε διαφορετικούς περιορισμούς, οι οποίοι συνεισφέρουν στην πολυπλοκότητα του προβλήματος.

3.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.

4.Σημαντικά χαρακτηριστικά των προβλημάτων διαφέρουν σε διαφορετικά περιβάλλοντα Επίσης, στο ίδιο περιβάλλον διαφέρουν από την ξεχωριστή περίπτωση αλγορίθμου επίλυσης.

5.Ανάλογα με το περιβάλλον ο χρόνος απόκρισης για τη δημιουργία χρονοπρογράμματος μπορεί να διαφέρει από μερικά δευτερόλεπτα έως κάποιες μέρες.

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

1.3.1 Χρονοπρογραμματισμός Σχολείου

Ο χρονοπρογραμματισμός σχολείου (***school timetabling***) αναφέρεται στο εβδομαδιαίο πρόγραμμα όλων των τάξεων ενός σχολείου με σκοπό την αποφυγή οι καθηγητές να διδάσκουν σε δύο τμήματα την ίδια χρονική στιγμή, και αντίστροφα. Αρχικά το μοντέλο αυτό, το οποίο αναφέρεται σαν μοντέλο «τάξη / καθηγητής», περιγράφεται σαν ένα απλοποιημένο πολυωνυμικό πρόβλημα που μπορεί να επιλυθεί σε πολυωνυμικό χρόνο, ενώ στη συνέχεια περιγράφεται σαν ένα βασικό πρόβλημα αναζήτησης και πρόβλημα βελτιστοποίησης ακολούθως.

1.3.2 Χρονοπρογραμματισμός Μαθημάτων Πανεπιστημίου

Ο χρονοπρογραμματισμός των μαθημάτων ενός πανεπιστημίου (***university course timetabling***) αναφέρεται στο εβδομαδιαίο πρόγραμμα όλων των διαλέξεων κάθε

μαθήματος, όταν δίνεται ο αριθμός των διαθέσιμων αιθουσών και των χρονικών περιόδων. Η κύρια διαφορά της κατηγορίας αυτής από τα προβλήματα που αφορούν τον χρονοπρογραμματισμό ενός σχολείου είναι ότι τα μαθήματα του πανεπιστημίου μπορούν να έχουν κοινούς φοιτητές, ενώ σε ένα σχολείο τα τμήματα είναι ανεξάρτητα σύνολα μαθητών. Συγκεκριμένα αν δύο μαθήματα έχουνε κοινούς φοιτητές, τότε αυτά «συγκρούονται» και δεν μπορούν να προγραμματιστούν την ίδια περίοδο. Επιπλέον οι καθηγητές των σχολείων συνήθως διδάσκουν σε περισσότερα από ένα τμήματα, ενώ ένας καθηγητής πανεπιστημίου μπορεί να διδάσκει ένα μόνο μάθημα. Σημαντικό παράγοντα επίσης σε ένα πρόβλημα χρονοπρογραμματισμού των μαθημάτων ενός πανεπιστημίου αποτελεί η διαθεσιμότητα των αιθουσών και χωρητικότητά τους, σε αντίθεση με το χρονοπρογραμματισμό ενός σχολείου όπου συνήθως σε κάθε αίθουσα αντιστοιχεί ένα τμήμα.

1.3.3 Χρονοπρογραμματισμός Εξετάσεων

Ο χρονοπρογραμματισμός εξετάσεων (***examination timetabling***) αναφέρεται στο πρόγραμμα εξετάσεων όλων των μαθημάτων ενός πανεπιστημίου για δοθέν χρονικό διάστημα. Η κατηγορία αυτή είναι όμοια με τον χρονοπρογραμματισμό των μαθημάτων ενός πανεπιστημίου, γεγονός που καθιστά δύσκολη τη διάκριση μεταξύ των δύο προβλημάτων.

1.4 Δυσκολίες Προβλημάτων Χρονοπρογραμματισμού

Ένα πρόβλημα timetabling μπορεί να αναπαρασταθεί σαν ένα σύνολο E γεγονότων, ένα σύνολο S σχισμών και ένα σύνολο C περιορισμών ανάμεσα στα γεγονότα. Στόχος είναι ο καθορισμός των γεγονότων σε σχισμές με τέτοιο τρόπο ώστε να ικανοποιούνται όλοι οι περιορισμοί. Συγκεκριμένα, τα προβλήματα παραγωγής ωρολογίου προγράμματος εξεταστικής περιόδου (***Examination Timetabling Problems- ETTP***) που θα εξεταστούν στη παρούσα εργασία, αναπαρίστανται από το σύνολο E των εξεταστέων μαθημάτων, το σύνολο S χρονοσχισμών ή περιόδων και το σύνολο C περιορισμών που σχετίζονται με την ανάθεση εξεταστέων μαθημάτων σε περιόδους. Με τον όρο χρονοσχισμή γίνεται αναφορά σε μία τιμή από το πεδίο τιμών της εκάστοτε μεταβλητής. Πιο

συγκεκριμένα, για τα προβλήματα κατασκευής ωρολογίου προγράμματος ο όρος αυτός αναφέρεται στο συνδυασμό των ημερών, ωρών και αιθουσών, οι οποίες όπως αναγράφεται στις γραμμές που ακολουθούν αποτελούν τις παραμέτρους που καθορίζουν το μέγεθος των πεδίων τιμών κάθε μεταβλητής. Τέτοιου είδους προβλήματα ανήκουν στην κλάση των προβλημάτων 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 παραμέτρους. Για τη δημιουργία του ωρολογίου προγράμματος εξεταστικής περιόδου λαμβάνουμε υπ' όψιν 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 30 μαθήματα προς εξέταση κατά τη διάρκεια 15 ημερών (3 εβδομάδων) με 4 τρίωρα και 5 αίθουσες διαθέσιμες κάθε μέρα, βλέπουμε ότι η πολυπλοκότητα του προβλήματος είναι $(15*4*5)^{30} = 300^{30}$. 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 υπολογιστικά με ακρίβεια.

1.5 Μέθοδοι Επίλυσης Προβλήματος

Δεδομένου ότι η ύπαρξη προβλημάτων χρονικού προγραμματισμού δεν είναι καθόλου ένα σύγχρονο ζήτημα, αλλά αντιθέτως προβληματίζει τους ερευνητές εδώ και δεκαετίες, είναι λογικό ότι αρχικά η προσέγγιση της λύσης γινόταν « με το χέρι», μία επίπονη διαδικασία που συχνά απαιτούσε την πολυήμερη εργασία αρκετών

ατόμων. Επιπροσθέτως, οι παράμετροι που έπρεπε να ελεγχθούν ήταν πολλές, και συχνά ήταν αρκετά σύνθετες. Έγινε λοιπόν αντιληπτό, ότι με τη χειρωνακτική προσέγγιση της λύσης, ελλοχεύει πάντα ο κίνδυνος η λύση που θα προκύψει να μην ικανοποιεί όλες τις παραμέτρους του προβλήματος, δεδομένου ότι δεν υπάρχει συστηματική διαδικασία για να ελεγχθεί μια χειρωνακτική λύση ως προς την πληρότητά της. Για παράδειγμα, είναι πιθανόν να προκύψουν λύσεις όπου σύμφωνα με το ωρολόγιο πρόγραμμα μαθημάτων ενός πανεπιστημίου ένας φοιτητής δε θα μπορεί να παρακολουθήσει τα μαθήματα που επιθυμεί γιατί ήταν προγραμματισμένα την ίδια ώρα.

Για τους παραπάνω λόγους, η προσοχή των ερευνητών εδώ και αρκετές δεκαετίες στράφηκε στον αυτοματοποιημένο χρονικό προγραμματισμό. Τα τελευταία σαράντα χρόνια πολλές μελέτες που σχετίζονται με τον αυτοματοποιημένο προγραμματισμό έχουν δημοσιευτεί, αρκετές εκ των οποίων έχουν εφαρμοσθεί με μεγάλη επιτυχία σε διάφορους οργανισμούς, ιδρύματα και άλλες επιχειρήσεις.

Είναι λοιπόν, προφανές ότι η πλειονότητα των πρώτων προσπαθειών αυτοματοποίησης της διαδικασίας που αναπτύχθηκαν για την επίλυση προβλημάτων χρονικού προγραμματισμού, βασίστηκαν στη μίμηση του χειρωνακτικού τρόπου επίλυσης του προβλήματος (υπολογισμός της λύσης «με το χέρι»). Όλες αυτές οι τεχνικές, που αποκαλούνται **άμεσα ευρετικές μέθοδοι** (***direct heuristics***) στηρίχθηκαν στη **διαδοχική επαύξηση του προβλήματος** (***successive augmentation***). Η μέθοδος αυτή χρησιμοποιούσε ένα ημιτελές ωρολόγιο πρόγραμμα και το επέκτεινε σταδιακά, προσθέτοντας ένα-ένα γεγονός (π.χ. διάλεξη), ώσπου όλα τα γεγονότα (π.χ. οι διαλέξεις) να ενταχθούν στο πρόγραμμα και να κατασκευαστεί τελικά ένα πλήρες ωρολόγιο πρόγραμμα. Η βασική ιδέα της προσέγγισης αυτής ήταν ότι έπρεπε να προγραμματιστεί η ώρα διεξαγωγής του γεγονότος που ήταν πιο δύσκολο να προγραμματιστεί (εμφάνιζε τους περισσότερους περιορισμούς). Η διαφορά των προσεγγίσεων αυτών κατά την εφαρμογή τους έγκειται στον ορισμό του τι σημαίνει ότι ένα γεγονός είναι δύσκολο να προγραμματιστεί.

Αργότερα, οι ερευνητές άρχισαν να εφαρμόζουν γενικές τεχνικές για την επίλυση των προβλημάτων χρονικού προγραμματισμού. Έτσι έκαναν την εμφάνισή τους προσεγγίσεις που βασιζόταν μεταξύ άλλων στον ακέραιο γραμμικό προγραμματισμό (***integer linear programming***), καθώς και στη ροή δικτύων (***network flow***). Στην εξέλιξη αυτή, αξίζει να σημειωθεί, ο ρόλος που διαδραμάτισε η αύξηση της ισχύος των υπολογιστών και η εξέλιξη της έρευνας στο χώρο των αλγορίθμων.

Επίσης, δεν ήταν λίγες οι περιπτώσεις που οι ερευνητές επιδίωξαν να διατυπώσουν σε μια διαφορετική βάση το πρόβλημα και να το αναγάγουν σε πρόβλημα που μπορεί να επιλυθεί σε μία μέθοδο, που είχε ήδη μελετηθεί αρκετά και ήταν περισσότερο κατανοητή, αυτή του χρωματισμού γραφημάτων (***graph coloring***).

Πρόσφατα την εμφάνιση τους στη βιβλιογραφία έκαναν με μεγάλη επιτυχία, κάποιες προσεγγίσεις βασισμένες σε τεχνικές αναζήτησης που χρησιμοποιούνται στον χώρο της Τεχνητής Νοημοσύνης (***Artificial Intelligence***). Μεταξύ άλλων έχουμε τους αλγόριθμους 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 διπλωματική εργασία θα ερευνήσουμε τεχνικές εύρεσης λύσης, δίνοντας έμφαση στις πιο πρόσφατες προσεγγίσεις και συγκεκριμένα στις προσεγγίσεις της τεχνητής νοημοσύνης.

1.6 Περιγραφή Προβλήματος Δημιουργίας Ωρολογίου Προγράμματος Εξεταστικής Περιόδου

Για προβλήματα κατασκευής ωρολογίου προγράμματος εξεταστικής περιόδου τελικός στόχος αποτελεί η ανάθεση μαθημάτων σε κατάλληλες ημέρες, ώρες και αίθουσες έτσι ώστε να ικανοποιούνται όλοι οι βασικοί περιορισμοί που αφορούν την ομαλή διεξαγωγή των εξετάσεων. Με άλλα λόγια πρόκειται για την ανάθεση ενός αριθμού εξεταστέων μαθημάτων σε έναν περιορισμένο αριθμό διαθέσιμων χρονικών περιόδων έτσι ώστε να ικανοποιείται ένα σύνολο από περιορισμούς. Οι περιορισμοί αυτοί μπορεί να διαφέρουν από πανεπιστήμιο σε πανεπιστήμιο ή ακόμη και από τμήμα σε τμήμα. Για παράδειγμα μερικά μαθήματα μπορεί να απαιτούν την χρήση ηλεκτρονικών υπολογιστών για την εξέτασή τους.

Οι περιορισμοί που εμπλέκονται συνήθως σε προβλήματα κατασκευής ωρολογίων προγραμμάτων χωρίζονται σε δύο κατηγορίες. Στην πρώτη κατηγορία ανήκουν οι **δεσμευτικοί (*hard*)** περιορισμοί και οι οποίοι αποτελούν βασική προϋπόθεση για την επίλυση του προβλήματος. Τέτοιου είδους περιορισμοί συνήθως

αναφέρονται σε λειτουργικούς περιορισμούς που δεν μπορούν να αγνοηθούν όταν είναι επιθυμητή μια πραγματική λύση. Για παράδειγμα ο περιορισμός που θέλει δύο μαθήματα να μην εξετάζονται ταυτόχρονα (ίδια ημέρα, ώρα και αίθουσα) ανήκει στην κατηγορία αυτή. Άλλος περιορισμός της ίδιας κατηγορίας είναι εκείνος που δεν επιτρέπει την εξέταση δύο μαθημάτων ίδιου εξαμήνου την ίδια ημέρα. Ένα ωρολόγιο πρόγραμμα που ικανοποιεί όλους τους δεσμευτικούς περιορισμούς ονομάζεται **αληθοφανής.**

Στην δεύτερη κατηγορία ανήκουν οι λεγόμενοι **επιθυμητοί (*soft*)** περιορισμοί. Πρόκειται για περιορισμούς οι οποίοι είναι δευτερεύουσας σημασίας. Ένα παράδειγμα επιθυμητού περιορισμού είναι η εξέταση συγκεκριμένων μαθημάτων μόνο τις Δευτέρες. Οι περιορισμοί αυτής της κατηγορίας είναι αυτοί που καθορίζουν την ποιότητα του προγράμματος. Η ικανοποίηση όλο και περισσότερων επιθυμητών περιορισμών αποτελεί μέτρο ποιότητας ενός προγράμματος. Καλής ποιότητας πρόγραμμα θεωρείται εκείνο το οποίο κατά πρώτον είναι αληθοφανές και δεύτερον ικανοποιεί τους επιθυμητούς περιορισμούς του σε ικανοποιητικό επίπεδο. Ουσιαστικά όμως η έννοια της ποιότητας είναι υποκειμενική και μπορεί να διαφέρει από πανεπιστήμιο σε πανεπιστήμιο. Για παράδειγμα, κάποιο πανεπιστήμιο επιθυμεί την μέρα-παρά-μέρα εξέταση των μαθημάτων (δεσμευτικός περιορισμός), ενώ για κάποιο άλλο που επιθυμεί μια πιο σύντομη εξεταστική περίοδο ο παραπάνω περιορισμός να αποτελεί μέρος των επιθυμητών περιορισμών.

1.7. Μαθηματικό Μοντέλο

Μια επίσημη προσέγγιση του προβλήματος βελτιστοποίησης κατασκευής ωρολογίου προγράμματος παρουσιάζεται παρακάτω και βασίζεται σε μοντέλο μαθηματικού προγραμματισμού. Έστω 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 ισχύσει ότι Y_{ik}=1 αν έχει ανατεθεί η τιμή k στη μεταβλητή i, διαφορετικά

Y_{ik}=0. Ακόμα, θα ισχύσει 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:

$$C = 0 \quad (1)$$

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

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

Εικόνα 1.1: Οι εξισώσεις του μαθηματικού μοντέλου του ωρολογίου προγράμματος.

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 δυνατόν περισσότερο. Η δεύτερη εξίσωση μας εξασφαλίζει ότι όλα τα μαθήματα έχουνε ανατεθεί σε ακριβώς μια χρονοθυρίδα το καθένα και τέλος η τρίτη εξίσωση διασφαλίζει ότι δεν παραβιάζεται κανένας περιορισμός.

ΚΕΦΑΛΑΙΟ 2

ΧΡΟΝΟΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΜΕ ΠΕΡΙΟΡΙΣΜΟΥΣ

2.1 Εισαγωγή

Ο χρονοπρογραμματισμός βασισμένος σε περιορισμούς (***constraint based scheduling***) αποτελεί μια προσέγγιση πραγματικών προβλημάτων χρονοπρογραμματισμού με τη χρήση περιορισμών στις μεταβλητές του προβλήματος. Η ανάπτυξη γενικών και εξειδικευμένων τεχνικών ικανοποίησης περιορισμών, καθιστούν το χρονοπρογραμματισμό με περιορισμούς πολύ δημοφιλή στην επίλυση συνδυαστικών προβλημάτων βελτιστοποίησης (***combinatorial optimization problems***) καθώς επιτυγχάνει γενικότητα, αποδοτικότητα και διαφάνεια των μεθόδων που χρησιμοποιεί.

Ένας περιορισμός (***constraint***) είναι μια λογική σχέση μεταξύ των διάφορων μεταβλητών του προβλήματος, σε κάθε μια από τις οποίες ανατίθεται μία τιμή από ένα δοθέν σύνολο τιμών. Κάθε περιορισμός επομένως περιορίζει τον αριθμό των πιθανών τιμών που κάθε μεταβλητή μπορεί να πάρει αντιπροσωπεύοντας έτσι μια μερική πληροφορία για τις μεταβλητές του προβλήματος. Ένα σημαντικό χαρακτηριστικό των περιορισμών είναι ο δηλωτικός τρόπος χρήσης τους, δηλαδή καθορίζουν τη σχέση μεταξύ των μεταβλητών δηλωτικά αποφεύγοντας τη χρήση οποιασδήποτε υπολογιστικής διαδικασίας για το σκοπό αυτό.

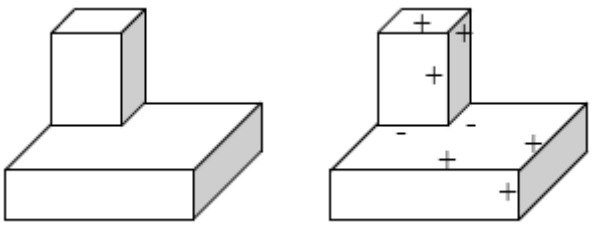
Γενικά τα προβλήματα χρονοπρογραμματισμού με περιορισμούς βασίζονται στη μοντελοποίηση και επίλυση πραγματικών προβλημάτων στα οποία ένα σύνολο δραστηριοτήτων πρέπει να καταναλώσει έναν περιορισμένο αριθμό πόρων (***resources***) και σε περιορισμένο χρονικό διάστημα. Ένα τέτοιο πρόβλημα περιλαμβάνει επομένως την κατανομή πόρων στις δραστηριότητες του προβλήματος και τη διάταξη δραστηριοτήτων σε κάθε περιορισμό.

Στις επόμενες ενότητες του κεφαλαίου θα παρουσιαστούν κάποια παραδείγματα από προβλήματα χρονοπρογραμματισμού με περιορισμούς μέσα από την περιοχή των προβλημάτων ικανοποίησης περιορισμών, τι γίνεται όταν τα

προβλήματα ικανοποίησης περιορισμών είναι σε πεπερασμένα ή μη πεδία, τα είδη των περιορισμών που μπορούμε να συναντήσουμε και τέλος τι γίνεται όταν χρησιμοποιούμε τοπική αναζήτηση στα προβλήματα ικανοποίησης περιορισμών.

2.2 Ιστορική Αναδρομή

Οι πρώτες ιδέες που οδήγησαν στον προγραμματισμό περιορισμών προέρχονται από το πεδίο της τεχνητής νοημοσύνης την δεκαετία του ‘60 και ‘70. Το πρόβλημα της ανάθεσης ετικετών σε σχήματα (***scene labeling***) αποτέλεσε το πρώτο πρόβλημα που μοντελοποιήθηκε ως πρόβλημα ικανοποίησης περιορισμών. Στόχος αυτού του προβλήματος αποτελεί η αναγνώριση τρισδιάστατων εικόνων με την ερμηνεία γραμμών σε δυσδιάστατα σχήματα. Με τον όρο γραμμή (***line***) αναφερόμαστε στην γραμμή ενός δυσδιάστατου σχήματος που ενώνει δύο σημεία αυτού. Μία μέθοδος παραγωγής τρισδιάστατης δομής με χρήση δυσδιάστατων σχημάτων αποτελεί η ανάθεση ετικετών στις γραμμές του δυσδιάστατου σχήματος. Οι ετικέτες τοποθετούνται στις γραμμές για την αναγνώριση της σχετικής θέσης αυτών στο χώρο. Υπάρχουν τέσσερις ετικέτες που χρησιμοποιούνται για τον χαρακτηρισμό των γραμμών. Μια γραμμή μπορεί να είναι κυρτή, κοίλη, ή να περικλείει το σχήμα είτε προς τα δεξιά είτε προς τα αριστερά. Η κυρτή γραμμή συμβολίζεται με το σύμβολο ‘-‘, η κοίλη γραμμή με το σύμβολο ‘+’ ενώ οι ακμές που περικλείουν το σχήμα συμβολίζονται με ένα κατευθυνόμενο τόξο ‘>’ ή ‘<’. Πρέπει να σημειωθεί ότι ο διαχωρισμός μεταξύ κοίλων και κυρτών γραμμών ποικίλλει ανάλογα με την σκοπιά που βλέπει ο παρατηρητής το αντικείμενο. Στη συνέχεια απεικονίζεται μια δυσδιάστατη εικόνα, όπου στις γραμμές της έχουν ανατεθεί ετικέτες των παραπάνω κατηγοριών (οι γραμμές που δεν εμφανίζεται ετικέτα είναι αυτές που περικλείουν το σχήμα και ο συμβολισμός τους είναι <).



Εικόνα 2.1: Δυσδιάστατη απεικόνιση με ετικέτες πάνω στις γραμμές.

Υπάρχουν πολλοί τρόποι που μπορούν να τοποθετηθούν οι ετικέτες σε μια εικόνα, αλλά μόνο μερικοί από αυτούς μπορούν να χαρακτηριστούν ότι απεικονίζουν μια τρισδιάστατη μορφή αυτής. Αυτό σημαίνει ότι ο μοναδικός περιορισμός προς ικανοποίηση είναι ότι δύο γραμμές που ενώνονται στα σημεία ένωσης της εικόνας θα πρέπει να έχουν την ίδια ετικέτα. Έτσι το πρόβλημα μειώνεται πολύ καθώς υπάρχει ένας περιορισμένος αριθμός νόμιμων ετικετών στα σημεία ένωσης. Από το παραπάνω σχήμα, οι επιτρεπόμενες ενώσεις ακμών είναι δύο:

Εικόνα 2.2: Οι δύο επιτρεπόμενες ενώσεις ακμών.

Μια άλλη εφαρμογή για περιορισμούς του Ivan Sutherland που αναπτύχθηκε στις αρχές της δεκαετίας του 1960 ήταν το πρωτοποριακό σύστημα Sketchpad, ένα σύστημα αλληλεπίδρασης γραφικών. Ο Sketchpad και ο απόγονός του ThingLab που δημιουργήθηκε από τον Alan Borning αποτελούσαν εφαρμογές αλληλεπίδρασης γραφικών οι οποίες επέτρεπαν στον χρήστη τον σχεδιασμό και τον χειρισμό περιορισμένων γεωμετρικών σχημάτων στην οθόνη του υπολογιστή. Αυτοί οι μέθοδοι συνέβαλαν στην ανάπτυξη μεθόδων τοπικής διάδοσης και constraint compiling.

Τα κυρίως βήματα στον προγραμματισμό περιορισμών πραγματοποιήθηκαν από τους Gallaire, Jaffer και Lassez οι οποίοι παρατήρησαν ότι ο λογικός προγραμματισμός ήταν ένα ιδιαίτερο είδος προγραμματισμού περιορισμών. Η δήλωση από πλευράς χρήστη για το τι πρέπει να λυθεί στο πρόβλημα και όχι το πώς θα λυθεί αυτό, αποτελεί τη βασική ιδέα του λογικού προγραμματισμού. Επίσης, ο κανόνας της ενοποίησης (***unification***) του λογικού προγραμματισμού μπορεί να χρησιμοποιηθεί από προβλήματα περιορισμών καθώς η διαδικασία που ακολουθεί ταιριάζει απόλυτα με τον τρόπο εύρεσης λύσης στα προβλήματα περιορισμών. Γρήγορα λοιπόν παρατηρείται ένα σύνολο κοινών στοιχείων μεταξύ του προγραμματισμού περιορισμών και του λογικού προγραμματισμού. Για αυτό το λόγο ο συνδυασμός των δύο παραπάνω μεθοδολογιών αποτέλεσε φυσική εξέλιξη καθώς ο λογικός προγραμματισμός αποδείχθηκε ιδανικό περιβάλλον για την επίλυση

προβλημάτων με περιορισμούς. Το γεγονός αυτό όμως δεν οδηγεί αναπόφευκτα στην ταύτιση των δύο εννοιών καθώς ο λογικός προγραμματισμός αποτελεί μόνο μια εκδοχή επίλυσης.

Ο σχεδιασμός, η κατασκευή ωρολογίων προγραμμάτων και η βελτιστοποίηση αποτελούν βασικές εφαρμογές των προβλημάτων περιορισμών. Για αυτό το λόγο το πεδίο της Επιχειρησιακής Έρευνας έστρεψε το ενδιαφέρον της στην επίλυση τέτοιου είδους προβλημάτων. Υπάρχει αλληλοεπικάλυψη των δύο πεδίων όσον αφορά τα NP-δύσκολα (***NP-hard***) συνδυαστικά προβλήματα. Ενώ η Επιχειρησιακή Έρευνα έχει μακρά παράδοση έρευνας αλλά και επιτυχημένων μεθόδων επίλυσης προβλημάτων χρησιμοποιώντας γραμμικό προγραμματισμό, ο προγραμματισμός περιορισμών δίνει έμφαση στην υψηλού επιπέδου μοντελοποίηση και μεθόδων επίλυσης που είναι εύκολοι στην κατανόηση από τον τελικό χρήστη. Η συνεργασία των δύο παραπάνω πεδίων αποτέλεσε ιδιαίτερα σημαντικό παράγονται στην εξέλιξη των προβλημάτων περιορισμών. Ο προγραμματισμός περιορισμών χρησιμοποιεί αλγορίθμους για την επίλυση των περιορισμών αυτών μέσα στους οποίους περιλαμβάνονται μερικοί που ανήκουν στο πεδίο της Επιχειρησιακής Έρευνας.

Όπως αναφέρθηκε στις προηγούμενες παραγράφους, ο προγραμματισμός περιορισμών συνδυάζει και αξιοποιεί ιδέες από διάφορα πεδία, όπως είναι: Τεχνητή Νοημοσύνη, Συνδυαστικοί Αλγόριθμοι, Υπολογιστική Λογική, Διακριτά Μαθηματικά, Επιχειρησιακή Έρευνα, Γλώσσες Προγραμματισμού και Συμβολική Υπολογιστική.

2.3 Προβλήματα Ικανοποίησης Περιορισμών

Ο Προγραμματισμός Με Περιορισμούς (***Constraint Programming ή CP***) αποτελεί ένα πλαίσιο εργασιών για την επίλυση συνδυαστικών προβλημάτων βελτιστοποίησης. Η βασική ιδέα στην οποία στηρίζεται είναι η μοντελοποίηση του προβλήματος σαν ένα σύνολο μεταβλητών, σε κάθε μια από τις οποίες αντιστοιχεί ένα πεδίο τιμών, και ένα σύνολο περιορισμών για τους πιθανούς συνδυασμούς τιμών των μεταβλητών. Καθώς τα πεδία τιμών είναι συνήθως πεπερασμένα, τα προβλήματα αυτά αναφέρονται σαν Προβλήματα Ικανοποίησης Περιορισμών (***Constraint Satisfaction Problems ή CSP***). Σκοπός των προβλημάτων αυτών είναι η εύρεση των τιμών των μεταβλητών, τέτοιων ώστε να ικανοποιούνται όλοι οι περιορισμοί του

προβλήματος. Σε μερικές περιπτώσεις επίσης χρησιμοποιείται και μία Αντικειμενική Συνάρτηση (***Objective Function***) στις μεταβλητές του προβλήματος και δηλώνει την ποιότητα της λύσης. Στις περιπτώσεις αυτές σκοπός είναι να βρεθεί μια ανάθεση τιμών στις μεταβλητές που να ελαχιστοποιούν ή να μεγιστοποιούν την αντικειμενική συνάρτηση. Τέτοια προβλήματα αναφέρονται γενικά ως Προβλήματα Βελτιστοποίησης Ικανοποίησης Περιορισμών (***Constraint Satisfaction Optimization Problems ή CSOP***).

Τα Προβλήματα Χρονοπρογραμματισμού (***Scheduling Problems***) ανήκουν στην περιοχή των συνδυαστικών προβλημάτων βελτιστοποίησης, επομένως μπορούν να περιγραφούν σαν Πρόβλημα Ικανοποίησης Περιορισμών (***CSP***). Για να μοντελοποιηθεί ένα πρόβλημα χρονοπρογραμματισμού σε Πρόβλημα Ικανοποίησης Περιορισμών πρέπει να αποφασιστεί πως θα απεικονιστούν τα αντικείμενα του προβλήματος στις μεταβλητές και τους περιορισμούς του προβλήματος CSP.

2.3.1 Παρουσίαση του Μοντέλου CSP

Ένα Πρόβλημα Ικανοποίηση Περιορισμών (***Constraint Satisfaction Problem ή CSP***) ορίζεται από ένα σύνολο μεταβλητών X_1, X_2, …, X_n, ένα σύνολο τιμών κάθε μία από τις οποίες ανατίθεται σε μία μεταβλητή και από ένα σύνολο περιορισμών, C_1, C_2, …, C_m, οι οποίοι περιγράφουν τη λογική σχέση μεταξύ των μεταβλητών. Κάθε μεταβλητή X_i έχει ένα κενό πεδίο (***domain***) D_i των δυνατών τιμών της. Κάθε περιορισμός C_i αναφέρεται σε κάποιο υποσύνολο των μεταβλητών και καθορίζει τους επιτρεπτούς συνδυασμούς τιμών γι’ αυτό το υποσύνολο. Μια κατάσταση του προβλήματος ορίζεται με ανάθεση τιμών σε μερικές ή όλες τις μεταβλητές, $\{X_i = v_i, X_j = v_j, \ldots\}$. Μια ανάθεση τιμής που δεν παραβιάζει κανέναν περιορισμό ονομάζεται Συνεπής (***Consistent***) ή νόμιμη ανάθεση. Πλήρης ανάθεση είναι μια ανάθεση που περιλαμβάνει όλες τις μεταβλητές, και λύση ενός προβλήματος ικανοποίησης περιορισμών είναι μια πλήρης ανάθεση τιμών που ικανοποιεί όλους τους περιορισμούς.

2.4. Παραδείγματα Προβλημάτων Ικανοποίησης Περιορισμών

2.4.1 Το Πρόβλημα του Χρωματισμού Χάρτη

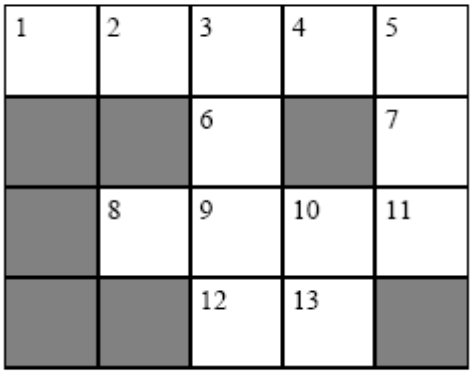
Το πρόβλημα του χρωματισμού χάρτη (***map coloring problem***) αποτελεί ένα από τα αρχέτυπα CSP προβλήματα. Το πρόβλημα αποτελείται από τον χρωματισμό διαφορετικών περιοχών ενός προκαθορισμένου χάρτη, με περιορισμένο αριθμό χρωμάτων, και υπόκειται στην περιοριστική συνθήκη ότι δύο παρακείμενες περιοχές δεν επιτρέπεται να έχουν το ίδιο χρώμα. Για παράδειγμα, ας θεωρήσουμε τον χάρτη της Αυστραλίας και ότι έχουμε τα χρώματα κόκκινο, κίτρινο και μπλε. Για κάθε περιοχή ορίζεται μία μεταβλητή WA, NT, SA, Q, NSW, V και T, στην οποία πρέπει να γίνει ανάθεση χρώματος. Κάθε μεταβλητή έχει ως πεδίο το σύνολο τιμών {κόκκινο, κίτρινο, μπλε}.

Εικόνα 2.3: Οι κύριες πολιτείες και περιφέρειες της Αυστραλίας. Ο χρωματισμός αυτού του χάρτη μπορεί να θεωρηθεί ως πρόβλημα ικανοποίησης περιορισμών.

Ο παρακάτω περιορισμός εγγυάται ότι κάθε ζεύγος παρακείμενων περιοχών δεν μπορεί να έχει το ίδιο χρώμα: $WA \neq NT \wedge WA \neq SA \wedge NT \neq SA \wedge NT \neq Q \wedge SA \neq Q \wedge SA \neq NSW \wedge SA \neq V \wedge Q \neq NSW \wedge NSW \neq V$

2.4.2 Το Πρόβλημα του Σταυρολέξου

Το πρόβλημα του σταυρολέξου έχει χρησιμοποιηθεί για την αξιολόγηση αλγορίθμων όσον αφορά προβλήματα ικανοποίησης περιορισμών. Το πρόβλημα ζητάει την τοποθέτηση λέξεων από το λεξικό (ή από κάποιο δοσμένο σύνολο λέξεων) είτε κάθετα είτε οριζόντια ανάλογα με τους συγκεκριμένους κάθε φορά περιορισμούς. Αν επιτρέπεται κάθε λέξη να τοποθετείται σε κάθε χώρο με το σωστό μέγεθος, μια πιθανή μορφοποίηση του προβλήματος σε γράφο περιορισμών φαίνεται στην παρακάτω εικόνα .



Εικόνα 2.4: Το πρόβλημα του σταυρόλεξου.

Υπάρχει μια μεταβλητή για κάθε τετράγωνο η οποία μπορεί να κρατήσει ένα χαρακτήρα. Το πεδίο τιμών κάθε μεταβλητής καθορίζεται από τα γράμματα της αλφαβήτου ενώ οι περιορισμοί καθορίζονται από την είσοδο των πιθανών λέξεων. Για παράδειγμα :

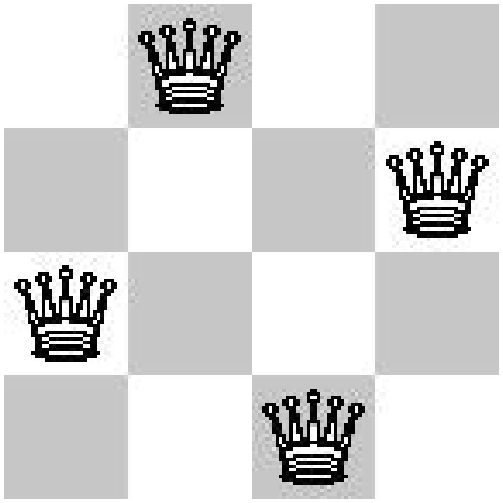
{HOSES, LASER, SHEET, SNAIL, ALSO, EARN, HIKE, IRON, SAME, EAT, LET, RUN, SUN, TEN, YES, BE, IT, NO, US}

Έτσι υπάρχει περιορισμός ανάμεσα στις μεταβλητές x_8, x_9, x_{10}, x_{11} οι οποίες επιτρέπουν να τους ανατεθούν λέξεις τεσσάρων γραμμάτων από την λίστα των λέξεων. Ο περιορισμός μπορεί να περιγραφεί από τη σχέση C = { (A, L, S, O), (E, A, R, N), (H, I, K, E), (I, R, O, N), (S, A, M, E) }. Αυτό σημαίνει ότι οι μεταβλητές x_8, x_9, x_{10}, x_{11} μπορούν να πάρουν μία από τις παραπάνω τιμές. Η λύση αυτού του

προβλήματος περιορισμών είναι η τοποθέτηση των γραμμάτων στα τετράγωνα με τέτοιο τρόπο ώστε μόνο οι επιτρεπτές λέξεις να εισαχθούν.

2.4.3. Το Πρόβλημα των N Βασιλισσών

Ένα ακόμα γνωστό CSP πρόβλημα είναι το πρόβλημα των N βασιλισσών (N-Queens). Στο πρόβλημα αυτό πρέπει να τοποθετηθούν N βασίλισσες πάνε σε μία σκακιέρα μεγέθους N × N έτσι ώστε όλες οι βασίλισσες να είναι σε διαφορετικές γραμμές, στήλες και διαγώνιους. Ας θεωρήσουμε το πρόβλημα των 4 βασιλισσών όπως φαίνεται στο παρακάτω σχήμα.



Εικόνα 2.5: Το πρόβλημα των 4 βασιλισσών σε σκακιέρα 4 x 4.

Μπορούμε να τυποποιήσουμε το πρόβλημα ως CSP θεωρώντας ότι κάθε βασίλισσα i έχει δύο μεταβλητές, την μεταβλητή Ri και Ci οι οποίες αντιστοιχούν στην ανάλογη γραμμή και στήλη που είναι τοποθετημένη η βασίλισσα πάνω στην σκακιέρα. Το πεδίο τιμών της κάθε μεταβλητής είναι {1,2,3,4}. Ο περιορισμός:

$R1 \neq R2 \wedge R1 \neq R3 \wedge R1 \neq R4 \wedge R2 \neq R3 \wedge R2 \neq R4 \wedge R3 \neq R4$,

εξασφαλίζει ότι δύο βασίλισσες δεν μπορούν να βρίσκονται στην ίδια γραμμή, ενώ ο περιορισμός:

$C1 \neq C2 \wedge C1 \neq C3 \wedge C1 \neq C4 \wedge C2 \neq C3 \wedge C2 \neq C4 \wedge C3 \neq C4$, εξασφαλίζει ότι δύο βασίλισσες δεν μπορούν να βρίσκονται στην ίδια στήλη, και τέλος οι περιορισμοί:

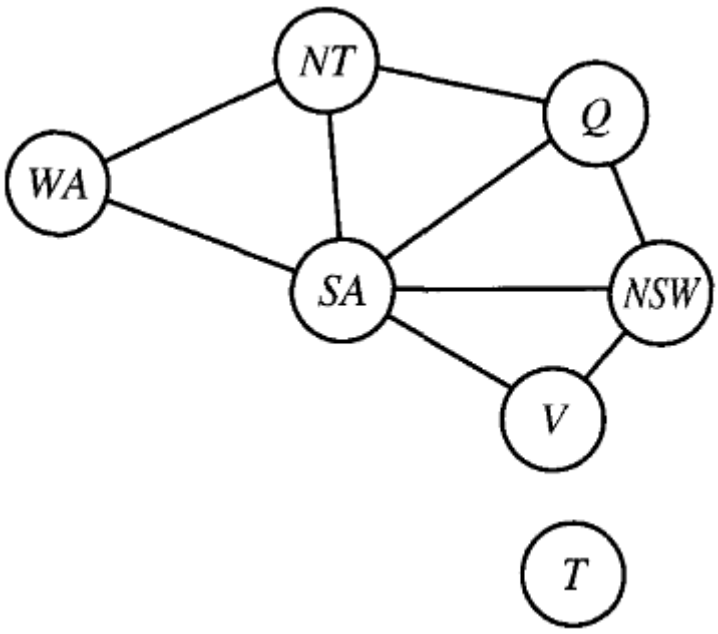
$C1 - R1 \neq C2 - R2 \wedge C1 - R1 \neq C3 - R3 \wedge C1 - R1 \neq C4 - R4 \wedge C2 - R2 \neq C3 - R3 \wedge C2 - R2 \neq C4 - R4 \wedge C3 - R3 \neq C4 - R4$

και

$C1 + R1 \neq C2 + R2 \wedge C1 + R1 \neq C3 + R3 \wedge C1 + R1 \neq C4 + R4 \wedge C2 + R2 \neq C3 + R3 \wedge C2 + R2 \neq C4 + R4 \wedge C3 + R3 \neq C4 + R4$, εξασφαλίζουν ότι δύο βασίλισσες δεν μπορούν να βρίσκονται στην ίδια διαγώνιο.

2.5. Μέθοδος Αναπαράστασης Περιορισμών

Μία από τις μεθόδους που έχουν αναπτυχθεί για την αναπαράσταση των περιορισμών του προβλήματος, είναι εκείνη με γράφο. Κάθε κόμβος του γράφου αντιστοιχεί σε μία μεταβλητή του προβλήματος, ενώ κάθε ακμή αντιστοιχεί σε περιορισμό. Χαρακτηριστικό παράδειγμα αναπαράστασης CSP με γράφο είναι το παράδειγμα χρωματισμού χάρτη που αναφέραμε.



Εικόνα 2.6: Το πρόβλημα του χρωματισμού του χάρτη σε αναπαράσταση ως γράφημα περιορισμών.

2.6. Προβλήματα Ικανοποίησης Περιορισμών σε Πεπερασμένα Πεδία

Το απλούστερο είδος CSP προβλημάτων είναι εκείνα που περιέχουν διακριτές μεταβλητές με πεπερασμένα πεδία τιμών. Χαρακτηριστικά παραδείγματα το παράδειγμα του χρωματισμού χάρτη και εκείνο των N βασιλισσών. Στην ομάδα των CSP με πεπερασμένα πεδία ανήκουν και τα προβλήματα εκείνα των οποίων οι μεταβλητές μπορούν να πάρουν τιμές true ή false.

Όπως είπαμε κάθε λύση του προβλήματος αποτελεί πλήρη ανάθεση τιμών σε όλες τις μεταβλητές. Εάν τώρα ο αριθμός των μεταβλητών είναι ίσως με n τότε κάθε λύση θα εμφανίζεται σε βάθος n του δέντρου αναζήτησης, γεγονός το οποίο κάνει τους αλγόριθμους αναζήτησης κατά βάθος πολύ δημοφιλείς.

Στην περίπτωση όπου το μέγιστο μέγεθος πεδίου οποιασδήποτε μεταβλητής του προβλήματος είναι d, τότε ο αριθμός όλων των πιθανών πλήρως αναθέσεων τιμών είναι O(dn), το οποίο είναι εκθετικό ως προς τον αριθμό των μεταβλητών. Αυτό έχει ως αποτέλεσμα όσο αυξάνεται ο αριθμός των μεταβλητών του προβλήματος τόσο να αυξάνεται εκθετικά και ο χώρος αναζήτησης. Στην χειρότερη περίπτωση μάλιστα δεν μπορούμε να περιμένουμε ότι ο χρόνος επίλυσης μπορεί να είναι μικρότερος του εκθετικού.

2.7. Προβλήματα Ικανοποίησης Περιορισμών σε Μη Πεπερασμένα Πεδία

Οι διακριτές μεταβλητές μπορούν επίσης να έχουν μη πεπερασμένα πεδία, για παράδειγμα το σύνολο των ακεραίων αριθμών ή το σύνολο των αλφαριθμητικών. Ας θεωρήσουμε ότι έχουμε δύο εργασίες J_1 και J_2 τις οποίες πρέπει να αναθέσουμε στο χρόνο. Εάν τώρα θεωρήσουμε ότι κάθε εργασία J_i έχει μία μεταβλητή $StartJ_i$ η οποία έχει ως πεδίο τιμών το σύνολο των ακεραίων κάθε ένας από τους οποίους αντιστοιχεί και σε μία ημέρα, τότε για τις μεταβλητές αυτές με μη πεπερασμένα πεδία είναι αδύνατο να εκφράσουμε περιορισμούς που να απαριθμούν όλους τους επιτρεπόμενους συνδυασμούς τιμών. Στην περίπτωση αυτή απαιτείται μία γλώσσα που να μπορεί να εκφράσει περιορισμούς. Εάν για παράδειγμα η εργασία J_1 απαιτεί 5 ημέρες για να διεκπεραιωθεί και ισχύει ότι πρέπει να προηγείται της εργασίας J_2, τότε

χρειαζόμαστε μία γλώσσα περιορισμών με αλγεβρικές ανισότητες όπως $StartJ_1 + 5 \leq StartJ_2$. Είναι πολύ απίθανο να λυθούν τέτοιου είδους περιορισμοί απαριθμώντας όλες τις πιθανές αναθέσεις τιμών καθώς υπάρχει άπειρος συνδυασμός από αυτές. Ειδικοί αλγόριθμοι αναζήτησης λύσης έχουν αναπτυχθεί για γραμμικούς περιορισμούς πάνω σε μεταβλητές ακεραίων πεδίων. Οι γραμμικοί περιορισμοί είναι περιορισμοί στους οποίους οι μεταβλητές εμφανίζονται μόνο σε γραμμική μορφή, για παράδειγμα X + Y = 1. Μπορεί να αποδειχθεί ότι δεν υπάρχει αλγόριθμος ο οποίος μπορεί να λύσει μη γραμμικούς περιορισμούς για μεταβλητές ακεραίων πεδίων. Σε κάποιες περιπτώσεις, μπορούμε να μετατρέψουμε ένα CSP με ακέραια πεδία σε πρόβλημα πεπερασμένων πεδίων οριοθετώντας τις τιμές όλων των μεταβλητών. Για παράδειγμα, σε ένα πρόβλημα χρονικού προγραμματισμού διεργασιών μπορούμε να θέσουμε ένα άνω όριο το οποίο θα ισούται με το άθροισμα της διάρκειας όλων των διεργασιών.

2.8. Περιορισμοί

Αφού εξετάσαμε τον τύπο των μεταβλητών που μπορούν να υπάρξουν σε ένα πρόβλημα CSP, ας δούμε τώρα και τον τύπο των περιορισμών.

2.8.1. Μοναδιαίος Περιορισμός (Unary Constraint)

Ο απλούστερος τύπος περιορισμού που υπάρχει είναι ο μοναδιαίος περιορισμός, ο οποίος περιορίζει την τιμή μίας μοναδικής μεταβλητής. Για παράδειγμα στο παράδειγμα του χρωματισμού γράφου μπορεί η μεταβλητή WA να μην επιτρέπεται να έχει το χρώμα κόκκινο. Κάθε μοναδιαίος περιορισμός μπορεί εξαλειφθεί απλά ελέγχοντας το πεδίο τιμών την αντίστοιχης μεταβλητής και αφαιρώντας όποια τιμή παραβιάζει τον περιορισμό.

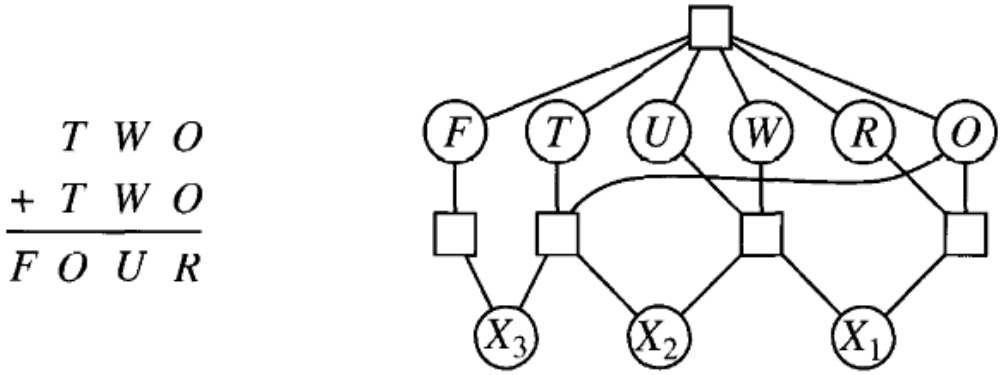
2.8.2. Δυαδικός Περιορισμός (Binary Constraint)

Ένας δυαδικός περιορισμός συσχετίζει τις τιμές δύο μεταβλητών, για παράδειγμα $X \neq Y$ αποτελεί ένα δυαδικό περιορισμό στον οποίο οι μεταβλητές X και

Υ δεν μπορούν να έχουν τις ίδιες τιμές. Οι αναπαράσταση δυαδικών περιορισμών μπορεί να γίνει με γράφο όπως είδαμε προηγουμένως.

2.8.3. Υψηλού Βαθμού Περιορισμός (High-order Constraint)

Οι Υψηλού βαθμού περιορισμοί συσχετίζουν τις τιμές τριών και περισσοτέρων μεταβλητών. Ένα οικείο παράδειγμα είναι αυτό του κρυπταριθμητικού αινίγματος.



Εικόνα 2.7: (α) Το κρυπταριθμητικό πρόβλημα. (β) Το υπεργράφημα των περιορισμών για το κρυπταριθμητικό πρόβλημα.

Στο πρόβλημα αυτό κάθε γράμμα αντιστοιχεί και σε ένα διαφορετικό ψηφίο. Το παραπάνω παράδειγμα μπορεί να παρασταθεί με έξι μεταβλητές οι οποίες περιορίζονται από τον περιορισμό alldiff (F, T, U, W, R, O) ο οποίος διασφαλίζει ότι κάθε γράμμα θα αντιστοιχεί και σε ένα διαφορετικό ψηφίο. Οι επιπλέον τέσσερις περιορισμοί για κάθε στήλη του αινίγματος που συσχετίζουν τις μεταβλητές μπορούν να εκφραστούν από τις σχέσεις :

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

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

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

$$X3 = F$$

όπου οι μεταβλητές X1, X2 και X3 είναι βοηθητικές μεταβλητές και αναπαριστούν το κρατούμενο 0 ή 1. Οι Υψηλού βαθμού περιορισμοί μπορούν να απεικονιστούν από ένα υπέρ-γράφο περιορισμών (***Constraint Hypergraph***) όπως αυτός της εικόνας 2.7(β), στον οποίο κάθε περιορισμός απεικονίζεται με ένα

τετράγωνο και οι ακμές οι οποίες συνδέουν τον περιορισμό με τις συσχετιζόμενες μεταβλητές.

2.8.4.Αυστηροί και Εύκαμπτοι Περιορισμοί (Hard and Soft Constraints)

Οι περιορισμοί που έχουμε περιγράψει μέχρι τώρα αποτελούν αυστηρούς περιορισμούς η παραβίαση των οποίων αποκλείει κάθε δυνητική λύση. Υπάρχουν όμως και περιπτώσεις πραγματικών εφαρμογών με CSP, οι οποίες μπορεί να περιέχουν περιορισμούς βάση των οποίων μία λύση μπορεί να θεωρηθεί καταλληλότερη σε σχέση με μία άλλη. Οι περιορισμοί αυτοί ονομάζονται εύκαμπτοι περιορισμοί και χρησιμοποιούνται κυρίως σε προβλήματα βελτιστοποίησης ή και τοπικής αναζήτησης. Οι Εύκαμπτοι περιορισμοί μπορούν να εκφραστούν χρησιμοποιώντας κάποια βάρη σε σχέση με τους αντίστοιχους περιορισμούς. Κάθε φορά που κάποιος περιορισμός παραβιάζεται, η αντίστοιχη τιμή βάρους προστίθεται στη συνάρτηση κόστους, βάση τις οποίας επιλέγεται ή απορρίπτεται αντίστοιχα μία λύση. Για παράδειγμα ας θεωρήσουμε το παρόν πρόβλημα κατάρτισης ωρολογίου προγράμματος, στο οποίο μία εξέταση παρακολουθείται από μεγάλο αριθμών φοιτητών και επιθυμούμε να την αναθέσουμε σε μία αίθουσα. Είναι λογικό ότι η αίθουσα την οποία θα επιλέξουμε, θα πρέπει να είναι μεγάλη παρά μικρή σε χωρητικότητα, αν και η ανάθεση της εξέτασης σε μικρή αίθουσα αποτελεί και αυτή λύση αλλά όχι την βέλτιστη. Έτσι θα μπορούσαμε να πούμε ότι η ανάθεση της εξέτασης σε μεγάλη αίθουσα θα είχε κόστος 0 ενώ η ανάθεση της εξέτασης σε μικρή αίθουσα θα είχε κόστος 1 προσδιορίζοντας με αυτόν τον τρόπο την βέλτιστη λύση.

2.9. Τοπική Αναζήτηση σε CSP

Οι αλγόριθμοι τοπική αναζήτησης αποδεικνύονται πολύ αποτελεσματικοί για την επίλυση πολλών προβλημάτων ικανοποίησης περιορισμών. Χρησιμοποιούν μία διατύπωση με πλήρεις καταστάσεις. Η αρχική κατάσταση αναθέτει τιμή σε κάθε μεταβλητή και η συνάρτηση διαδοχών συνήθως αναλαμβάνει να αλλάζει την τιμή μίας-μίας μεταβλητής. Στο πρόβλημα των βασιλισσών, Εικόνα 2.5**,** η αρχική κατάσταση θα μπορούσε να είναι μια τυχαία διάταξη των 4 βασιλισσών σε 4 στήλες

και η συνάρτηση διαδοχών παίρνει μία βασίλισσα και εξετάζει κινήσεις της σε άλλες θέσεις της στήλης.

Για την επιλογή νέας τιμής για μια μεταβλητή, ο πιο συνήθης ευρετικός μηχανισμός είναι να επιλέγεται η τιμή που προκύπτει με τον ελάχιστο αριθμό συγκρούσεων με άλλες μεταβλητές, δηλαδή τον ευρετικό μηχανισμό των **ελάχιστων συγκρούσεων**(***min-conflicts***). Ο ψευδοκώδικας του αλγορίθμου παρουσιάζεται στην Εικόνα 2.8:

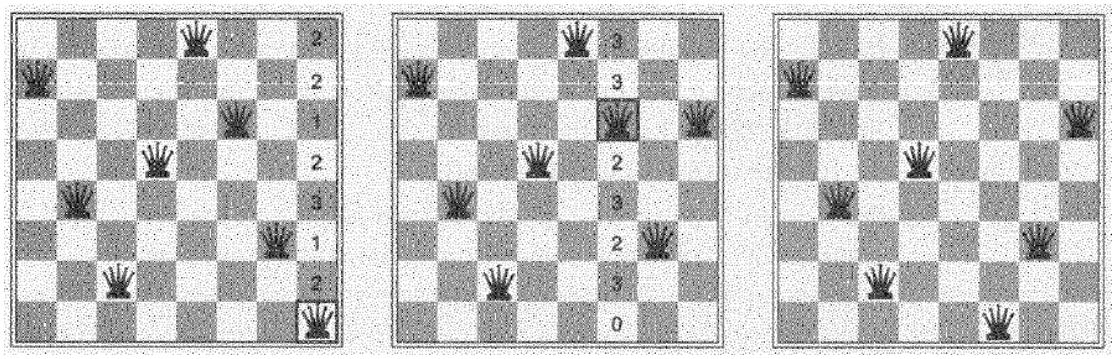
function MIN-CONFLICTS(csp, max_steps) **returns** a solution or failure
  inputs: csp, a constraint satisfaction problem
    max_steps, the number of steps allowed before giving up

  $current \leftarrow$ an initial complete assignment for csp
  for i = 1 to max_steps **do**
    if $current$ is a solution for csp **then return** $current$
    $var \leftarrow$ a randomly chosen, conflicted variable from VARIABLES[csp]
    $value \leftarrow$ the value v for var that minimizes CONFLICTS($var, v, current, csp$)
    set $var = value$ in $current$
  return *failure*

Εικόνα 2.8: Ο αλγόριθμος των ελάχιστων συγκρούσεων (MIN-CONFLICTS).

Η αρχική κατάσταση μπορεί να επιλέγεται τυχαία ή με μία διαδικασία άπληστης ανάθεσης τιμών που επιλέγει μια τιμή ελάχιστων συγκρούσεων για την κάθε μεταβλητή με τη σειρά. Η συνάρτηση CONFLICTS καταμετρά τον αριθμό των περιορισμών που παραβιάζονται από μια συγκεκριμένη τιμή, με διαδεδομένη την υπόλοιπη τρέχουσα ανάθεση τιμών.

Ακολουθεί ένα παράδειγμα για την λύση ενός προβλήματος 8 βασιλισσών σε δύο κινήσεις χρησιμοποιώντας την μέθοδο των ελάχιστων συγκρούσεων.



Εικόνα 2.9: Εφαρμογή της μεθόδου των ελάχιστων συγκρούσεων για ένα πρόβλημα 8 βασιλισσών.

Σε κάθε στάδιο επιλέγεται μία βασίλισσα για νέα ανάθεση τιμής πάνω στη στήλη της. Ο αριθμός των συγκρούσεων (στη συγκεκριμένη περίπτωση ο αριθμός των βασιλισσών που απειλούνται) εμφανίζεται σε κάθε τετράγωνο. Ο αλγόριθμος μετακινεί τη βασίλισσα στο τετράγωνο των ελάχιστων συγκρούσεων, αίροντας τις ισοδυναμίες τυχαία.

Η μέθοδος των ελάχιστων συγκρούσεων είναι εκπληκτικά αποτελεσματική για πολλά προβλήματα ικανοποίησης περιορισμών, ιδιαίτερα όταν του δοθεί μια λογική αρχική κατάσταση. Μία από τις εντυπωσιακές ιδιότητες του αλγορίθμου είναι πως στο πρόβλημα των n-βασιλισσών ο χρόνος εκτέλεσης της μεθόδου των ελάχιστων συγκρούσεων είναι σχεδόν ανεξάρτητος από το μέγεθος του προβλήματος.

Άλλο ένα πλεονέκτημα της τοπικής αναζήτησης είναι ότι μπορεί να χρησιμοποιείται σε online περιβάλλοντα όπου το πρόβλημα αλλάζει. Αυτό είναι ιδιαίτερα σημαντικό στα προβλήματα χρονοπρογραμματισμού. Για παράδειγμα ένα εβδομαδιαίος χρονοπρογραμματισμός αεροπορικών πτήσεων μπορεί να περιλαμβάνει χιλιάδες πτήσεις και δεκάδες χιλιάδες τοποθετήσεις προσωπικού, αλλά η κακοκαιρία σε ένα αεροδρόμιο μπορεί να κάνει τον χρονοπρογραμματισμό ανέφικτο, γι αυτό θα θέλαμε χρονοπρογραμματισμό με έναν ελάχιστο αριθμό αλλαγών. Αυτό μπορεί να γίνει εύκολα με έναν αλγόριθμο τοπικής αναζήτησης που ξεκινά από τον τρέχοντα χρονοπρογραμματισμό.

ΚΕΦΑΛΑΙΟ 3

ΜΟΝΤΕΛΟΠΟΙΗΣΗ ΠΡΟΓΡΑΜΜΑΤΟΣ ΕΞΕΤΑΣΤΙΚΗΣ ΩΣ CSP

3.1. Εισαγωγή

Εξετάζουμε το πρόβλημα χρονοπρογραμματισμού εξετάσεων το οποίο προκύπτει στα πανεπιστήμια που ακολουθούν ένα μεγάλης κλίμακας εκπαιδευτικό πρόγραμμα, όπου κάθε φοιτητής επιλέγει ένα εξατομικευμένο πακέτο εξετάσεων από ένα ευρύ διατμηματικό σύνολο ενοτήτων, πολλές από τις οποίες βρίσκονται εκτός του δεδομένου τμήματος. Τα γεγονότα εδώ είναι τα εξεταζόμενα μαθήματα και οι υποδοχείς χρόνου είναι οι πιθανές ώρες έναρξης των εκάστοτε εξεταζόμενων μαθημάτων.

Βασικός στόχος 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 περισσότερο γίνεται στην πραγματικότητα. Επομένως, οι μοντελοποιήσεις που μπορεί να έχει ένα πρόγραμμα ποικίλουν ανάλογα με το 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 θέλει να προσομοιωθεί η πραγματικότητα τόσο πιο πολύπλοκη θα είναι η κωδικοποίηση του προβλήματος σε μια γλώσσα προγραμματισμού από τον ίδιο τον προγραμματιστή. Έτσι βλέπουμε συχνά να γίνονται κάποιες παραδοχές κατά τη διάρκεια της κωδικοποίησης για να διευκολυνθεί σε ένα βαθμό η διαδικασία μοντελοποίησης.

Το πρόβλημα αυτό είναι NP-Complete και διασκορπισμένο με τοπικά ελάχιστα που κάνουν ιδιαιτέρως δύσκολη την εφαρμογή ευρετικών τεχνικών αναζήτησης. Η πολυπλοκότητά του γίνεται αισθητή και από το μέγεθος του διαστήματος των λύσεων. Για παράδειγμα εάν υπάρχουν x πιθανές ώρες έναρξης και y εξεταζόμενα μαθήματα τότε προκύπτουν x^y πιθανά προγράμματα. Στο

συγκεκριμένο πρόβλημα που μελετάμε, του τμήματος Μηχανικών Πληροφορικής και Τηλεπικοινωνιών στην Κοζάνη για το ακαδημαϊκό έτος 2011/2012, με 36 εξεταζόμενα μαθήματα και 60 υποδοχείς χρόνου (από 4 ημερησίως για 15 μέρες) θα είναι 60^{36} τα πιθανά προγράμματα. Αυτός ο αριθμός αντιπροσωπεύει το πλήρες σύνολο των πιθανών λύσεων, πολλές από τις οποίες δε θα λαμβάνονταν υπ' όψιν από έναν άνθρωπο που θα έκανε με το χέρι το πρόγραμμα αυτό.

Στη συνέχεια του κεφαλαίου θα γίνει αρχικά μια περιγραφή του προβλήματος και θα δοθεί η δομή του. Επίσης θα αναλυθούν οι δομές δεδομένων που θα χρησιμοποιηθούν για να αναπαρασταθεί το πρόβλημα καθώς και οι αυστηροί και εύκαμπτοι περιορισμοί που χρησιμοποιήθηκαν για την μοντελοποίησή του. Τέλος, θα αναφερθούν και μερικές πιθανές επεκτάσεις και βελτιώσεις του προβλήματος.

3.2. Περιγραφή του προβλήματος

Σκοπός του προβλήματος κατασκευής προγράμματος εξεταστικής περιόδου είναι η ανάθεση των μαθημάτων κάθε ακαδημαϊκού εξαμήνου σε μέρα, ώρα και αίθουσα. Η υλοποίηση αυτή γίνεται μέσω περιορισμών. Οι περιορισμοί που είναι αυστηροί πρέπει οπωσδήποτε να ικανοποιηθούν ώστε να βρεθεί λύση στο πρόβλημα. Είναι λειτουργικοί περιορισμοί που δεν πρέπει να αγνοηθούν όταν είναι επιθυμητή μια λύση που να ανταποκρίνεται στην πραγματικότητα. Από την άλλη, οι εύκαμπτοι περιορισμοί δεν είναι να απαραίτητο να ικανοποιηθούν ώστε να υπάρξει λύση αλλά είναι αυτοί οι περιορισμοί που κάνουν το πρόγραμμα ποιοτικότερο.

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 περαιτέρω και καθένα από αυτά αντιστοιχείται στην εξέταση ενός μαθήματος. Ακόμα υπάρχουν πέντε συνολικά διαθέσιμες αίθουσες για την διεξαγωγή της εξεταστικής. Κάθε μάθημα διδάσκεται από ένα μόνο καθηγητή. Άλλο χαρακτηριστικό του μαθήματος είναι το εξάμηνο στο οποίο διδάσκεται. Η παράμετρος του εξαμήνου βοηθάει στο να μην εξετάζονται ποτέ μαθήματα του ίδιου εξαμήνου την ίδια μέρα. Επίσης, δε μπορούν να εξεταστούν δύο διαφορετικά μαθήματα την ίδια μέρα, την ίδια ώρα και στην ίδια αίθουσα. Μαθήματα του ίδιου καθηγητή δεν γίνεται να εξετάζονται ταυτόχρονα. Μαθήματα που ανήκουν στο ίδιο εξάμηνο δεν πρέπει να συνωστίζονται αλλά να υπάρχουν κενές μέρες ενδιάμεσά τους.

Μαθήματα όπου έχουνε δηλωθεί προς εξέταση από μεγάλο αριθμό φοιτητών δεν δύναται να πραγματοποιούνται σε αίθουσες μικρής χωρητικότητας γιατί δημιουργείται πρόβλημα υπερπληθυσμού. Τέλος, θα πρέπει να εξετάζονται δύο έως τέσσερα το πολύ μαθήματα σε μία μέρα έτσι ώστε να μην προκύψουν μέρες όπου εξετάζονται πολλά μαθήματα ή μέρες όπου δεν εξετάζεται κανένα μάθημα, θα πρέπει δηλαδή να υπάρχει μια ισόποση κατανομή των μαθημάτων με τις μέρες.

Η δομή του προβλήματος καθιστά το πρόγραμμα αρκετά σύνθετο. Έτσι, παρουσιάστηκε 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 αρχικά εξετάστηκαν για το ποια θα επιλεγεί ήταν η συναρτησιακή (δηλαδή με χρήση συναρτήσεων) και ο αντικειμενοστραφής προγραμματισμός. Το πλεονέκτημα της συναρτησιακής μεθόδου είναι η απλότητά της. Στην, αντίπερα όχθη, τα οφέλη που παίρνουμε εάν χρησιμοποιήσουμε την αντικειμενοστραφή μέθοδο είναι αρκετά και επειδή έχει επιλεχθεί η χρήση μιας αντικειμενοστραφούς γλώσσας, οδηγηθήκαμε στη χρήση αυτής της μεθόδου. Όπως θα φανεί στη συνέχεια, το γεγονός αυτό θα επηρεάσει πολύ τις δομές δεδομένων που θα χρησιμοποιήσουμε.

3.2.1. Δεδομένα

Αρχικά, θα αναφέρουμε ότι η «κύρια» μεταβλητή του προγράμματος είναι το μάθημα. Παρουσιάστηκε, όμως, το εξής πρόβλημα. Υπάρχουνε αρκετά στοιχεία που θα πρέπει να αντιστοιχηθούν σε κάθε μάθημα (η μέρα εξέτασης, η ώρα εξέτασης, η αίθουσα εξέτασης, ο αριθμός των φοιτητών που έχουν επιλέξει το μάθημα για να τον παρακολουθήσουν και επομένως να εξεταστούν σε αυτό, το όνομα του καθηγητή που διδάσκει το μάθημα, η ημέρα που προτιμάει ο καθηγητής να εξεταστεί το μάθημα που διδάσκει, το εξάμηνο στο οποίο διδάσκεται, αν το μάθημα εμπλέκεται σε σύγκρουση ή όχι). Μερικά από αυτά τα στοιχεία, όμως, όπως οι ημέρες, οι ώρες και οι αίθουσες εξέτασης πρέπει να μεταβάλλονται συνεχώς. Έτσι, δημιουργήσαμε δύο διαφορετικές κλάσεις.

3.2.1.1. Κλάση Plirofories

Στην πρώτη κλάση, η οποία ονομάστηκε ***Plirofories***, θα περιέχονται η ημέρα εξέτασης του μαθήματος, η ώρα εξέτασης του, η αίθουσα στην οποία θα

πραγματοποιηθεί η εξέταση και τέλος αν το μάθημα εμπλέκεται σε σύγκρουση ή όχι. Οι μεταβλητές αυτές έχουν συγκεκριμένα πεδία τιμών. Δηλαδή:

- Η μεταβλητή ***mera***, η οποία αναφέρεται στην ημέρα εξέτασης, είναι τύπου int (δηλαδή ακέραιου τύπου) και θα παίρνει τιμές από 0 έως 14 επειδή έχουμε ορίσει ότι η εξεταστική περίοδος διαρκεί συνολικά 15 ημέρες.
- Η μεταβλητή ***ora***, η οποία αντιστοιχεί στην ώρα εξέτασης του μαθήματος, είναι τύπου int (δηλαδή ακέραιου τύπου) και θα παίρνει τιμές από 0 έως 3 αφού έχουμε ορίσει 4 ακαδημαϊκά τρίωρα εξέτασης των μαθημάτων.
- Η μεταβλητή ***aithousa,*** η οποία αναφέρεται στην αίθουσα εξέτασης του μαθήματος, είναι τύπου int (δηλαδή ακέραιου τύπου) και θα παίρνει τιμές από 0 έως 4 επειδή έχουμε ορίσει 5 διαθέσιμες αίθουσες εξέτασης.
- Τέλος, η μεταβλητή ***sugrousi***, η οποία αντιστοιχεί στην περίπτωση που το μάθημα εμπλέκεται σε σύγκρουση ή όχι, είναι τύπου int (δηλαδή ακέραιου τύπου) και θα παίρνει τιμές 0 ή 1.

Ας δούμε ένα παράδειγμα για να γίνει περισσότερο κατανοητή η μορφή της κλάσης ***Plirofories***. Έστω ότι ο αλγόριθμος του προβλήματος κατά την εκτέλεση του προγράμματος έδωσε για ένα μάθημα τις τιμές ***mera***=7, ***ora***=3, ***aithousa***=2, ***sugrousi***=0. Αυτό σημαίνει ότι το συγκεκριμένο μάθημα θα εξεταστεί την ημέρα 7 (δηλαδή την Τετάρτη της δεύτερης βδομάδας της εξεταστικής περιόδου), την ώρα 3 (δηλαδή από τις 15:00 έως τις 18:00) και στην αίθουσα 2 (δηλαδή στην αίθουσα Β). Επειδή η μεταβλητή ***sugrousi*** είναι 0 αυτό σημαίνει ότι το συγκεκριμένο μάθημα δεν εμπλέκεται σε σύγκρουση, δηλαδή η συγκεκριμένη ανάθεση τιμών δεν παραβιάζει κανέναν περιορισμό.

Αναφέρεται επίσης ότι οι τιμές των μεταβλητών αυτών δε δίνονται από τον χρήστη αλλά από τον αλγόριθμο του προβλήματος κατά την διάρκεια εκτέλεσης του προγράμματος.

3.2.1.2. Κλάση Stoixeia

Στην δεύτερη κλάση, η οποία ονομάστηκε ***Stoixeia***, θα περιέχονται το όνομα του μαθήματος, το όνομα του καθηγητή, ο κωδικός του καθηγητή που

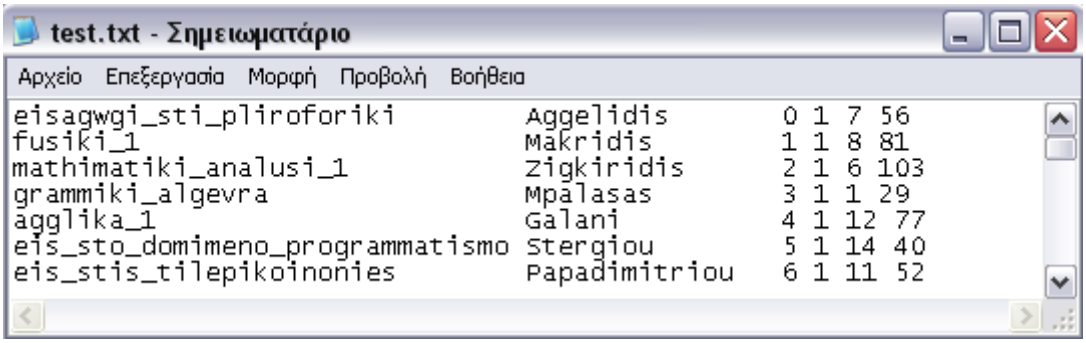
διδάσκει το μάθημα, το εξάμηνο κατά το οποίο διδάσκεται το μάθημα, η μέρα που προτιμάει ο καθηγητής να εξεταστεί το μάθημα και ο αριθμός των φοιτητών που έχουν δηλώσει το μάθημα και συνεπώς θα εξεταστούν σε αυτό. Οι μεταβλητές αυτές παίρνουν σαν πεδία τιμών τα εξής:

- Η μεταβλητή ***onoma_math***, η οποία αναφέρεται στο όνομα του μαθήματος, είναι τύπου string (δηλαδή είναι συμβολοσειρά) και επομένως μπορεί να αποτελείται από κάθε συνδυασμό χαρακτήρων.
- Η μεταβλητή ***onoma_kath***, η οποία αναφέρεται στο όνομα του καθηγητή που διδάσκει το μάθημα, είναι τύπου string (δηλαδή είναι συμβολοσειρά) και επομένως μπορεί να αποτελείται από κάθε συνδυασμό χαρακτήρων.
- Η μεταβλητή ***kodikos_kath***, η οποία αντιστοιχείται στον κωδικό του καθηγητή, είναι τύπου int (δηλαδή ακεραίου τύπου) και θα παίρνει θετικές τιμές. Για ευκολία στην υλοποίηση του προγράμματος επιλέξαμε στον κώδικα να αναφερόμαστε στους καθηγητές με βάση ένα μοναδικό κωδικό για τον καθένα, αφού η χρήση και η επεξεργασία αριθμών είναι πιο εύκολη από αυτή των συμβολοσειρών.
- Η μεταβλητή ***eks,*** η οποία αναφέρεται στο εξάμηνο που διδάσκεται το μάθημα, είναι τύπου int (δηλαδή ακεραίου τύπου) και παίρνει θετικές τιμές.
- Η μεταβλητή ***protimisi***, η οποία αναφέρεται στη μέρα που προτιμάει ο καθηγητής να εξεταστεί το μάθημα, είναι τύπου int (δηλαδή ακεραίου τύπου) και παίρνει τιμές από 0 έως 14, όπως ακριβώς και η μεταβλητή ***mera*.**
- Η μεταβλητή ***arithmos_foititwn***, η οποία αναφέρεται στον αριθμό των φοιτητών που έχουν δηλώσει το μάθημα και συνεπώς θα εξεταστούν σε αυτό, είναι τύπου int (δηλαδή ακεραίου τύπου) και παίρνει θετικές τιμές.

Ας δούμε άλλο ένα παράδειγμα για να κατανοήσουμε και τη μορφή της κλάσης ***Stoixeia***. Έστω ότι έχουμε τις τιμές ***onoma_math***=eisagwgi_stin_pliroforiki, ***onoma_kath***=Aggelidis, ***kodikos_kath***=0, ***eks***=1, ***protimisi***=7, ***arithmos_foititwn***=56. Αυτό σημαίνει ότι το μάθημα Εισαγωγή στη Πληροφορική το διδάσκει ο καθηγητής Αγγελίδης, ο οποίος έχει κωδικό 0. Επίσης, ότι το μάθημα ανήκει στο πρώτο εξάμηνο, ο καθηγητής Αγγελίδης προτιμά να εξεταστεί το μάθημα την ημέρα

7,δηλαδή την Τετάρτη της δεύτερης εβδομάδας και οι φοιτητές που έχουν δηλώσει το μάθημα προς εξέταση είναι 56.

Αξίζει να αναφερθεί ότι οι μεταβλητές της κλάσης ***Stoixeia*** (σε αντίθεση με τις μεταβλητές της κλάσης ***Plirofories*** οι οποίες δίνονται από τον αλγόριθμο του προβλήματος) δίνονται από τον χρήστη μέσω ενός αρχείου .txt το οποίο διαβάζει το πρόγραμμα ως είσοδο στα αρχικά βήματα εκτέλεσής του.



Εικόνα 3.1: Οι μεταβλητές της κλάσης Stoixeia που δίνονται από το χρήστη μέσω του αρχείου test.txt ως είσοδο στο πρόγραμμα. (από αριστερά προς τα δεξιά: onoma_math, onoma_kath, kodikos_kath, eks, protimisi, arithmos_foititwn).

Τα αντικείμενα των κλάσεων αυτών εισάγονται σε δύο πίνακες γραμμές οι οποίοι ονομάζονται ***Arhikos_pinakas*** και ***Lista*** και θα έχουν τύπους δεδομένων ***Plirofories*** και ***Stoixeia*** αντίστοιχα. Το μέγεθός των πινάκων θα είναι ίσο με το συνολικό αριθμό μαθημάτων. Συγκεκριμένα:

- Arhikos_ pinakas = new Plirofories[num];
- Lista = new Stoixeia[num];

1	2	3	.....	Num
A1	A2	A3	.....	ANum
↓				↓
(7, 3, 2, 0)				(13, 1, 4, 0)

Πίνακας Α: Η δομή δεδομένων του πίνακα Arhikos_pinakas.

Οι τιμές του πίνακα ***Arhikos_ pinakas*** αλλάζουν συνεχώς κατά τη διάρκεια εκτέλεσης του αλγορίθμου ανάλογα με τους περιορισμούς. Στο τέλος της εκτέλεσης, οι τιμές που θα είναι στα κελιά του πίνακα θα είναι και η λύση που θα επιστρέφει ο αλγόριθμος. Θα αντιπροσωπεύει δηλαδή το τελικό ωρολόγιο πρόγραμμα εξεταστικής.

1	2	3	.....	Num
S1	S2	S3	.....	SNum
↓				↓
(Eisagwgi_stin_Pliroforiki, Aggelidis, 0, 1, 7, 56)				(Fusiki_1, Makridis, 1, 1, 8, 81)

Πίνακας Β: Η δομή δεδομένων του πίνακα Lista.

Στη δική μας προσέγγιση ως κριτήριο λύσης για τον αλγόριθμο χρησιμοποιήθηκε ένας μετρητής με όνομα ***count_sugrousewn*** και ο οποίος «κρατάει» τον συνολικό αριθμό συγκρούσεων. Στόχος είναι ο μηδενισμός αυτού του μετρητή, δηλαδή ο μηδενισμός των συγκρούσεων, μέσω διαδοχικών αναθέσεων τιμών στις μεταβλητές του προγράμματος. Για να προκύψει η τιμή της μεταβλητής ***count_sugrousewn*** γίνεται έλεγχος των περιορισμών οι οποίοι θα αναλυθούν στη συνέχεια. Μόλις βρεθεί κάποιος περιορισμός που παραβιάζεται, η ***count_sugrousewn*** αυξάνεται κατά ένα και η τιμή της μεταβλητής ***sugrousi*** για το συγκεκριμένο μάθημα στον πίνακα ***Arhikos_pinakas*** γίνεται ίση με ένα (αρχικά έχουν αρχικοποιηθεί όλες στο μηδέν). Επομένως το πρόγραμμα γνωρίζει ανά πάσα στιγμή αν υπάρχουν συγκρούσεις και πόσες είναι αυτές. Έτσι μπορεί να αλλάξει τις μεταβλητές που είναι υπεύθυνες για τις συγκρούσεις, μέχρι να μην παραβιάζεται κανένας περιορισμός και συνεπώς να έχουμε μηδέν συγκρούσεις, ώστε το πρόγραμμα να εξάγει μια λύση η οποία θα είναι και το τελικό ωρολόγιο πρόγραμμα εξεταστικής.

3.3. Περιορισμοί

Όπως αναφέραμε και σε προηγούμενη ενότητα του κεφαλαίου, υπάρχουν δύο τύποι περιορισμών. Οι αυστηροί περιορισμοί, οι οποίοι πρέπει να ικανοποιηθούν με κάθε τρόπο έτσι ώστε να υπάρξει λύση και οι εύκαμπτοι περιορισμοί, οι οποίοι μπορεί να ικανοποιούνται αλλά δεν είναι αναγκαίο για το τελικό αποτέλεσμα. Στην

ενότητα αυτή θα γίνει πλήρης αναφορά καθένα περιορισμού και πως αυτός υλοποιείται μέσω του κώδικα.

Όλοι οι περιορισμοί είναι μεγαλύτερης τάξεως γιατί εμπλέκονται πολλές ιδιότητες του εκάστοτε μαθήματος κάθε φορά. Όλοι οι περιορισμοί εκτελούνται με διαδοχικά if. Αν οι συνθήκες των if δεν ικανοποιούνται τότε το πρόγραμμα προχωρά παρακάτω χωρίς να εκτελέσει αυτά τα if αφού δεν θα έχουμε σύγκρουση. Αντιθέτως, αν οι συνθήκες ικανοποιούνται τότε θα υπάρξει σύγκρουση. Έχουμε υποθέσει ότι τα μαθήματα που θα ελεγχθούν αν ικανοποιούν τους περιορισμούς είναι τα i και j.

3.3.1. Αυστηροί Περιορισμοί

Οι αυστηροί περιορισμοί είναι:

- Τα μαθήματα που ανήκουν στο ίδιο εξάμηνο δεν εξετάζονται την ίδια μέρα. Κωδικοποιείται ως εξής:

```
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);
                        }
}
```

- Δύο μαθήματα δεν είναι δυνατό να εξετάζονται την ίδια μέρα, ώρα και στην ίδια αίθουσα. Κωδικοποιείται 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()) {
                        count_sugkrousewn ++;
                        Arhikos_pinakas[i].set_sugrousi(1);
     }
   }
}
```

- Δύο μαθήματα που διδάσκονται από τον ίδιο καθηγητή δεν γίνεται να εξεταστούν την ίδια μέρα και ώρα. Κωδικοποιείται 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()) {
                    count_sugkrousewn ++;
                    Arhikos_pinakas[i].set_sugrousi(1);
    }
  }
}
```

- Για να αποφύγουμε την εξέταση μαθημάτων που ανήκουν στο ίδιο εξάμηνο να εξετάζονται μέσα σε σύντομο χρονικό διάστημα χρησιμοποιήσαμε τον ακόλουθο περιορισμό, ο οποίος ελέγχει αν δύο μαθήματα εξετάζονται σε διαδοχικές μέρες. Αν ναι, τότε ο μετρητής με όνομα ***count_apostasewn*** αυξάνεται κατά ένα. Όταν ο μετρητής ξεπεράσει την τιμή τρία τότε έχουμε σύγκρουση. Έτσι αποφεύγεται να εξεταστούν πάνω από τρία μαθήματα του ίδιου εξαμήνου σε διαδοχικές μέρες. Ο περιορισμός αυτός κωδικοποιείται 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) {
                count_sugkrousewn ++;
                Arhikos_pinakas[i].set_sugrousi(1);
        }
  }
```

```
}
```

- Για να αποφύγουμε μέρες όπου εξετάζονται λίγα μαθήματα ή κανένα δημιουργήσαμε έναν περιορισμό όπου σε κάθε μέρα θα εξετάζονται τουλάχιστον δύο μαθήματα. Για την κωδικοποίηση χρειαστήκαμε ένα μετρητή, με όνομα ***count_mathimaton***, ο οποίος έχει μέγεθος ίσο με 15 (τόσες είναι οι μέρες που διαρκεί η εξεταστική) και θα μετρά τον αριθμό των μαθημάτων που εξετάζονται κάθε μέρα. Η κωδικοποίηση αυτού του περιορισμού είναι η εξής:

```
for(int i=0;i<15;i++) {
        if (count_mathimaton[i] < 2) {
                count_sugkrousewn ++;
                Arhikos_pinakas[i].set_sugrousi(1);
        }
}
```

- Για να αποφύγουμε μέρες όπου εξετάζονται πολλά μαθήματα δημιουργήσαμε έναν περιορισμό όπου σε κάθε μέρα θα εξετάζονται το πολύ τέσσερα μαθήματα. Το σκεπτικό είναι παρόμοιο με αυτό του προηγουμένου περιορισμού. Η κωδικοποίηση είναι η εξής:

```
for(int i=0;i<15;i++) {
    if (count_mathimaton[i] > 4) {
        count_sugkrousewn ++;
        Arhikos_pinakas[i].set_sugrousi(1);
     }
}
```

- Τέλος, ο αριθμός των φοιτητών που εξετάζονται σε κάποιο μάθημα δεν πρέπει να είναι μεγαλύτερος από την χωρητικότητα της αίθουσας. Η κωδικοποίηση είναι η εξής:

```
if(Lista[i].get_arithmos_foititwn()>xwr_aithousas[Arhikos_pinakas[i].get_aithousa()]) {
                count_sugkrousewn ++;
```

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

3.3.2. Εύκαμπτοι Περιορισμοί

Οι εύκαμπτοι περιορισμοί είναι:

- Όπως έχουμε αναφέρει παραπάνω υπάρχει μία μεταβλητή (***protimisi***) που αναφέρεται στη μέρα που προτιμά ο καθηγητής να εξεταστεί το μάθημα. Δημιουργήσαμε λοιπόν ένα περιορισμό ο οποίος μετρά πόσες μέρες διαφορά υπάρχει ανάμεσα στη μέρα που προτιμά ο καθηγητής και στη μέρα που είχαμε ορίσει αρχικά να εξεταστεί το μάθημα και το ονομάσαμε ***kostos***. Σκοπός μας είναι να μειωθεί όσο το δυνατό περισσότερο το ***kostos*** έτσι ώστε αν είναι εφικτό η μέρα που προτιμά ο καθηγητής να εξεταστεί το μάθημα να είναι και η μέρα που θα βγάλει στο τέλος για λύση το πρόγραμμα. Η κωδικοποίηση είναι η εξής:

```
if (Arhikos_pinakas[i].get_mera() == Lista[i].get_protimisi() ) {
                kostos[i]=0;
 }
else if (Arhikos_pinakas[i].get_mera() > Lista[i].get_protimisi() ) {
                kostos[i]=(Arhikos_pinakas[i].get_mera() - Lista[i].get_protimisi() );
}
else if (Lista[i].get_protimisi() > Arhikos_pinakas[i].get_mera() ) {
               kostos[i]=(Lista[i].get_protimisi() - Arhikos_pinakas[i].get_mera() );
}
```

3.4. Συζήτηση

Παρόλο που το σύστημα βρίσκει χρονοδιαγράμματα πολύ καλά, μερικές βελτιώσεις και επεκτάσεις είναι πιθανές. Δηλαδή:

Πρόσθετες σκέψεις για τις αίθουσες: Κατά την περίπτωση που υπάρχουν μεγάλες σε χωρητικότητα αίθουσες, τότε θα μπορούσε δύο εξετάσεις να διεξαχθούν στο ίδιο μέρος την ίδια χρονική στιγμή αν βέβαια κανένας φοιτητής δεν χρειάζεται να εξεταστεί και στα δύο αυτά μαθήματα.

Εξετάσεις με διαφορετική διάρκεια: Ο αλγόριθμος για την επίλυση του προβλήματος χρονοπρογραμματισμού μπορεί να χρησιμοποιηθεί για ένα μεγάλο εύρος προβλημάτων εξετάσεων, με μια κατάλληλη μετατροπή δεδομένων. Παρόλα αυτά το πλαίσιο εργασίας όπως παρουσιάστηκε μέχρι τώρα υποθέτει ότι όλες οι εξετάσεις έχουν την ίδια ακριβώς χρονική διάρκεια (3 ώρες). Σε μερικές περιπτώσεις οι φοιτητές ενδέχεται να έχουνε μερικές εξετάσεις των δύο ωρών για παράδειγμα.

ΚΕΦΑΛΑΙΟ 4

ΑΝΑΛΥΣΗ ΑΛΓΟΡΙΘΜΟΥ

4.1. Εισαγωγή

Στο κεφάλαιο αυτό αναλύεται ο αλγόριθμος τοπικής αναζήτησης που αναπτύχθηκε για το πρόβλημα κατασκευής του προγράμματος εξεταστικής για το Πανεπιστήμιο Δυτικής Μακεδονίας και συγκεκριμένα για το τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών. Θα γίνει μια λεπτομερής περιγραφή και ανάλυση του αλγορίθμου. Θα δοθεί επίσης ο ψευδοκώδικας του, ο κώδικας σε C++ όπως υλοποιήθηκε στο πρόγραμμά μας καθώς και ο τρόπος διαχείρισης των δομών δεδομένων που αναπτύχθηκαν στο προηγούμενο κεφάλαιο.

Στο δικό μας πρόβλημα όπως επίσης και σε πολλά άλλα προβλήματα αναζήτησης, δεν μας ενδιαφέρει η διαδρομή προς μία κατάσταση στόχου αλλά η ίδια η κατάσταση του στόχου. Στις περιπτώσεις αυτές μπορούμε να χρησιμοποιήσουμε μια τεχνική που ονομάζεται **επαναληπτική βελτίωση** (***iterative improvement***). Στην τεχνική αυτή ξεκινάμε με μια μοναδική τρέχουσα κατάσταση και προσπαθούμε να τη βελτιώσουμε. Το ίδιο πλαίσιο εργασίας μπορεί να εφαρμοστεί και σε προβλήματα βελτιστοποίησης όπου μας ενδιαφέρει η λύση που βελτιστοποιεί μια δοσμένη αντικειμενική συνάρτηση. Χαρακτηριστικό των αλγόριθμων τοπικής αναζήτησης είναι ότι ανήκουν στην κατηγορία των ευρετικών μηχανισμών και χρησιμοποιούν την τεχνική της επαναληπτικής βελτίωσης.

Έτσι, ο αλγόριθμος που χρησιμοποιήθηκε για την κατασκευή του προγράμματος εξεταστικής είναι αλγόριθμος τοπικής αναζήτησης και συγκεκριμένα 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

Αν και η Τεχνητή Νοημοσύνη γνώριζε και αντιμετώπιζε Προβλήματα Ικανοποίησης Περιορισμών για πολλά χρόνια, δεν ήταν εξελιγμένη μέχρι τις αρχές της δεκαετίας του 1990 όταν η διαδικασία για την επίλυση των μεγάλων CSP κωδικοποιήθηκε σε αλγοριθμική μορφή. Αρχικά, ο Mark Johnston του Επιστημονικού Ινστιτούτου Διαστημικού Τηλεσκοπίου έψαξε για μια μέθοδο για να προγραμματίσει τις αστρονομικές παρατηρήσεις με το διαστημικό τηλεσκόπιο Hubble. Στο πλαίσιο των πειραμάτων αυτών, δημιούργησε ένα νευρωνικό δίκτυο ικανό να λύσει ένα πρόβλημα N-βασιλισσών (για 1024 βασίλισσες). Ο Steven Minton και ο Andy Philips ανέλυσαν τον αλγόριθμο νευρωνικού δικτύου και τον διαχώρισαν σε δύο φάσεις. Η πρώτη φάση ήταν μια αρχική ανάθεση χρησιμοποιώντας ένα άπληστο αλγόριθμο και η δεύτερη ήταν η φάση ελαχιστοποίησης των συγκρούσεων (που αργότερα ονομάστηκε «Min-conflicts»).

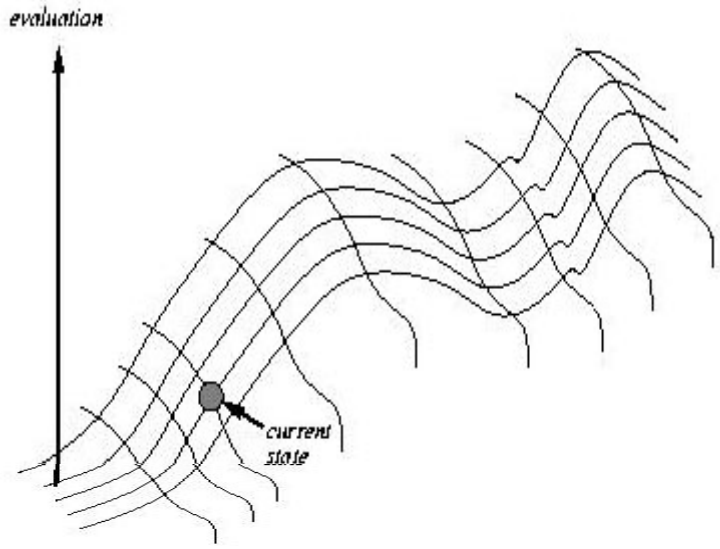
Έχει δειχτεί ότι ο συγκεκριμένος αλγόριθμος μπορεί να εφαρμοστεί και να λύσει εύκολα το πρόβλημα των 1.000.000 βασιλισσών σε μέσο όρο πενήντα βήματα και το πρόβλημα των 3.000.000 βασιλισσών σε λιγότερο από ένα λεπτό. Ο αλγόριθμος ελάχιστων συγκρούσεων έχει χρησιμοποιηθεί ευρέως σε προβλήματα περιορισμών, και ειδικότερα σε προβλήματα timetabling . Η επιτυχία του οφείλεται στην μεγάλη ταχύτητα για επίλυση και στην απλή φιλοσοφία του.

Στο πρόγραμμά μας θα χρησιμοποιήσουμε τον αλγόριθμο ελάχιστων συγκρούσεων με τυχαίες επαναλήψεις και όχι τον απλό αλγόριθμο. Η διαφορά τους είναι ότι ο πρώτος μπορεί να κάνει επανεκκίνηση στο πρόβλημα που εξετάζει ενώ ο δεύτερος όχι, γεγονός που κάνει που τον κάνει ακόμα καταλληλότερο στην επίλυση του προβλήματός μας.

4.2.2. Γενική Ανάλυση του Αλγορίθμου

Ο αλγόριθμος ελάχιστων συγκρούσεων με τυχαίες επανεκκινήσεις χρησιμοποιεί, όπως έχουμε προαναφέρει, την τεχνική της επαναληπτικής βελτίωσης. Ας δούμε τώρα ένα-ένα τα βήματα του τρόπου λειτουργίας του:

- Διαλέγουμε μια «λύση» από το χώρο αναζήτησης και την αποτιμούμε. Η λύση αυτή ονομάζεται τρέχουσα.
- Εφαρμόζουμε έναν μετασχηματισμό στην τρέχουσα λύση με σκοπό να παράγουμε μία νέα λύση και την οποία στη συνέχεια αξιολογούμε.
- Αν η νέα λύση είναι καλύτερη από την τρέχουσα τότε την αντικαθιστούμε με την τρέχουσα λύση, αλλιώς απορρίπτουμε τη νέα λύση.
- Επαναλαμβάνουμε τα παραπάνω βήματα μέχρι κανένας μετασχηματισμός να μην βελτιώνει περαιτέρω την τρέχουσα λύση.



Εικόνα 4.1: Γράφημα για τον τρόπο λειτουργίας του αλγορίθμου στο τοπίο του χώρου καταστάσεων.

Ο αλγόριθμος αυτός λειτουργεί βελτιώνοντας μια μοναδική τρέχουσα κατάσταση και γενικά μετακινείται μόνο σε γειτονικές της καταστάσεις. Σε κάθε βήμα του αλγορίθμου έχουμε μια πλήρη αλλά ατελή λύση στο δοσμένο πρόβλημα αναζήτησης. Μπορεί να μην έχουμε βρει την βέλτιστη λύση αλλά έχουμε βρει μια «αρκετά καλή» λύση σε πολύ μικρό χρονικό διάστημα. Αυτή είναι άλλωστε και μία

από της σημαντικότερες ιδιότητες του αλγορίθμου, να μπορεί δηλαδή να βρει ικανοποιητικές λύσεις σε μεγάλους ή και άπειρους χώρους λύσεων.

4.2.2.1. Ανάλυση του Αλγορίθμου στο πρόβλημα εξεταστικής

Αφού αναλύσαμε τα βήματα που εκτελεί ο αλγόριθμος γενικά, ας δούμε τώρα πως «μεταφράζονται» αυτά τα βήματα στο δικό μας πρόβλημα χρονοπρογραμματισμού της εξεταστικής.

Αρχικά να αναφέρουμε ότι ο αλγόριθμος Min-conflicts with random restarts δέχεται σαν είσοδο ένα πρόβλημα ικανοποίησης περιορισμών και επιστρέφει σαν έξοδο την λύση του προβλήματος αυτού, διαφορετικά επιστρέφει αποτυχία. Πιο συγκεκριμένα, οι είσοδοι που δέχεται είναι το πρόβλημα ικανοποίησης περιορισμών (***CSP***), ένας μέγιστος αριθμός βημάτων (***max_epanalipseis*** θα ονομάζεται η μεταβλητή αυτή στον κώδικά) όπου όταν ξεπεραστεί αυτός ο αριθμός ο αλγόριθμος θα σταματάει την αναζήτηση λύσης και τέλος ένας μέγιστος αριθμός επανεκκινήσεων (***max_epanekkiniseis*** η ονομασία του στον κώδικα) που επιτρέπεται να τρέξει ο αλγόριθμος πριν «παραιτηθεί» από την προσπάθεια για επίλυση και θα επιστρέφει αποτυχία. Η διαδικασία που θα εκτελεί κάθε φορά ο αλγόριθμος στο πρόγραμμα αναλύεται μέσω δέκα βημάτων και είναι η εξής:

1. Γίνεται μια τυχαία ανάθεση τιμών σε όλες τις μεταβλητές του προβλήματος.
2. Γίνεται έλεγχος συγκρούσεων και καταγράφεται ο συνολικός αριθμός συγκρούσεων.
3. Ο αλγόριθμος μπαίνει σε ένα βρόγχο με όριο βημάτων τον αριθμό των μέγιστων επαναλήψεων.
4. Εάν με την τυχαία ανάθεση τιμών που δόθηκε δεν υπάρχουν συγκρούσεις, επιστρέφεται η ανάθεση αυτή ως λύση και ο αλγόριθμος τερματίζεται.
5. Εάν η τυχαία αρχική ανάθεση τιμών δεν αποτελεί λύση (δηλαδή έχουμε συγκρούσεις), 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. Αφού έχουν τελειώσει όλες οι αναθέσεις, δίνεται στη μεταβλητή η τιμή που μειώνει τον αριθμό των συγκρούσεων περισσότερο από τις υπόλοιπες.
8. Τα βήματα 4 έως 7 επαναλαμβάνονται μέχρι να μην υπάρχουν συγκρούσεις ή να 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 βήματα 1 έως 8. Η ίδια διαδικασία επαναλαμβάνεται μέχρι 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 εμφανίζεται ως λύση διαφορετικά επιστρέφει αποτυχία.

4.2.2.2 Ψευδοκώδικας αλγορίθμου

Ο ψευδοκώδικας του αλγορίθμου που περιγράφει τον τρόπο λειτουργίας του στο πρόβλημα χρονοπρογραμματισμού εξεταστικής 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

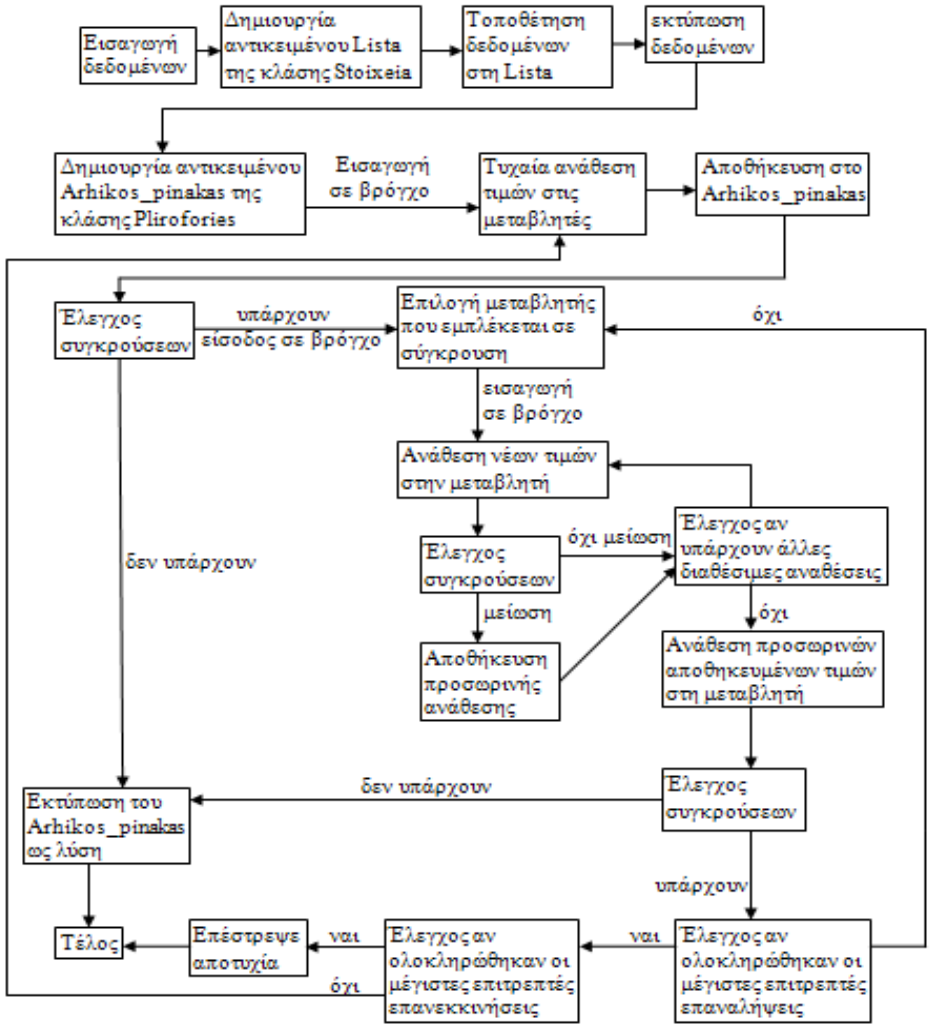
επέστρεψε αποτυχία

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.

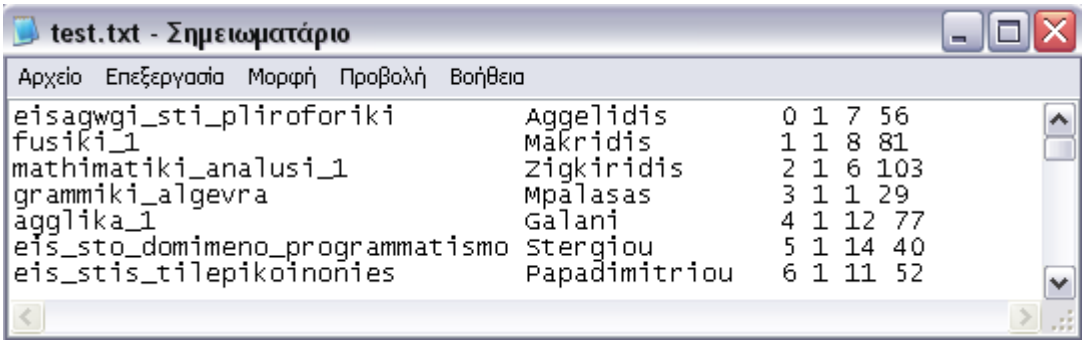
4.2.3 Διαχείριση Δομών Δεδομένων από Min-conflicts

Ας δούμε τώρα την κίνηση του αλγορίθμου κατά την προσπάθεια εύρεσης λύσης, παράλληλα με τις δομές δεδομένων. Έχουμε αναλύσει λεπτομερώς σε προηγούμενο κεφάλαιο (κεφάλαιο 3, ενότητα 3.2.1) ότι οι δομές δεδομένων που έχουμε είναι οι Plirofories, Stoixeia, Lista και Arhikos_pinakas. Η κίνηση του αλγορίθμου φαίνεται στην παρακάτω εικόνα.



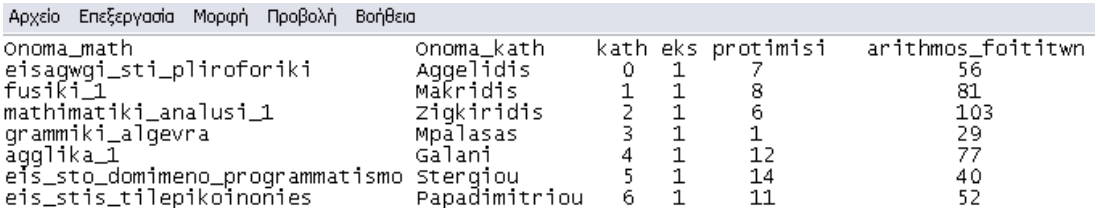
Εικόνα 4.2: Κίνηση αλγορίθμου παράλληλα με δομές δεδομένων.

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, το εξάμηνο κατά το οποίο διδάσκεται το μάθημα, τη μέρα που προτιμάει ο καθηγητής να εξεταστεί το μάθημα και πόσοι φοιτητές έχουν εγγραφεί στο μάθημα αυτό. Τα στοιχεία αυτά δίνονται ως είσοδο στο πρόγραμμα μέσω ενός αρχείου .txt και έχουν τη παρακάτω μορφή:



Εικόνα 4.3: Τα στοιχεία κάθε μαθήματος που δίνονται από το χρήστη μέσω του αρχείου test.txt ως είσοδο στο πρόγραμμα.(από αριστερά προς τα δεξιά: onoma_math, onoma_kath, kodikos_kath, eks, protimisi, arithmos_foititwn).

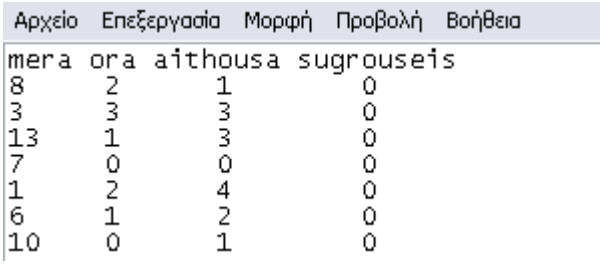
Στη συνέχεια, τα στοιχεία του κάθε μαθήματος τοποθετούνται στη δομή Lista όπου κάθε γραμμή αντιστοιχεί σε ένα μάθημα. Δηλαδή, για τον παραπάνω πίνακα το πρώτο μάθημα θα έχει όνομα ***Eisagwgi stin pliroforiki***, θα διδάσκεται από τον καθηγητή ***Aggelidi*** με κωδικό ***0***, θα είναι μάθημα του ***$1^{ου}$*** εξαμήνου, ο καθηγητής θα προτιμά να εξεταστεί την ***$7^{η}$*** μέρα από τις συνολικά 15 μέρες της εξεταστικής περιόδου και τέλος ο αριθμός των φοιτητών που έχουν εγγραφεί στο μάθημα και θα εξεταστούν είναι ***56***. 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:



Εικόνα 4.4: Η μορφή του πίνακα Lista.

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 1 (2η 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 την παρακάτω μορφή:



Εικόνα 4.5: Η μορφή του πίνακα Arhikos_pinakas.

Παρατηρούμε ότι στη στήλη ***sugrousi*** αρχικά όλες οι τιμές είναι στο 0. Αυτό γίνεται γιατί αρχικά έχουμε υποθέσει ότι κανένα μάθημα δεν εμπλέκεται σε σύγκρουση.

Η επόμενη κίνηση του αλγορίθμου είναι να κάνει έλεγχο συγκρούσεων. Ελέγχει όλους τους περιορισμούς για όλες τις μεταβλητές ώστε να εντοπίσει τον συνολικό αριθμό παραβιάσεων περιορισμών (συγκρούσεις) καθώς και εκείνες τις μεταβλητές οι οποίες εμπλέκονται σε κάποια παραβίαση. Στο τέλος του ελέγχου, η τιμή του πεδίου ***sugrousi*** 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, ο αλγόριθμος επιστρέφει την δομή Arhikos_pinakas σαν λύση του προβλήματος, εκτυπώνει τα αποτελέσματα και τερματίζεται. Αντίθετα, εάν εντοπίσει κάποιο αριθμό συγκρούσεων, εισέρχεται σε ένα βρόγχο ο οποίος τερματίζεται όταν δεν εντοπίζονται πλέον συγκρούσεις ή όταν ολοκληρωθούν οι μέγιστες επιτρεπτές επαναλήψεις (max_epanalipseis). Όσο βρίσκεται στο βρόγχο αυτό, πραγματοποιεί τα ακόλουθα βήματα:

1. Αρχικά επιλέγει τυχαία κάποια από τις μεταβλητές που εμπλέκονται σε σύγκρουση με βάση την τιμή του πεδίου sugrousi της καθεμίας.

2. 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.
3. Έπειτα αναθέτει στη μεταβλητή που επέλεξε, τις τιμές που έχουν κρατηθεί στις προσωρινές μεταβλητές καθώς για αυτές ο αριθμός των συγκρούσεων έχει μειωθεί περισσότερο.
4. 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.

Όταν εξέλθει ο αλγόριθμος από το βρόγχο, εάν ο αριθμός συγκρούσεων έχει πάρει την τιμή 0, τότε ο αλγόριθμος επιστρέφει την δομή Arhikos_pinakas ως λύση του προβλήματος, εκτυπώνει τα αποτελέσματα και τερματίζεται.

Εάν έχει ολοκληρωθεί ο μέγιστος αριθμός επαναλήψεων, ο αλγόριθμος απέτυχε να βρει λύση. Στη συνέχεια ελέγχει αν έχει ολοκληρωθεί ο μέγιστος αριθμός επανεκκινήσεων. Αν ναι, τότε επιστρέφει αποτυχία και τερματίζεται. Αν όχι, εκτελείται από την αρχή μέχρι να βρει λύση ή να ολοκληρωθούν οι μέγιστες δυνατές επανεκκινήσεις.

4.2.4 Η κωδικοποίηση του αλγορίθμου στο πρόγραμμα

Στο σημείο αυτό θα δείξουμε τη διαδικασία, που εκτελεί ο αλγόριθμος στην προσπάθειά του να βρει λύση, βήμα-βήμα στον κώδικά μας.

1. Αρχικά, όσο ο αλγόριθμος δεν βρίσκει μηδέν συγκρούσεις, δηλαδή όσο δε βρίσκει λύση και όσο δεν έχουν ξεπεραστεί οι μέγιστες επαναλήψεις (50) θα επιλέγει μια τυχαία τιμή από το 0 έως το num (num είναι το σύνολο το μαθημάτων). Όσο η τιμή αυτή δεν αντιστοιχεί σε μάθημα που εμπλέκεται σε σύγκρουση, θα επιλέγεται μια καινούρια τιμή συνέχεια μέχρι να βρεθεί τιμή που

να αντιστοιχεί σε μάθημα που εμπλέκεται σε σύγκρουση ή μέχρι να ολοκληρωθεί ένας μέγιστος αριθμός επαναλήψεων (1000). Παρακάτω φαίνεται η κωδικοποίηση:

```
while ((max_epanalipseis<50) && (min_sugrouseis!=0)) {
        do {
            temp = (rand()%num);
            epoxes1++;
        }
        while ((Arhikos_pinakas[temp].get_sugrousi() != 1) && (epoxes1<1000));
```

2. Στη συνέχεια ξαναγίνονται όλες οι τιμές σύγκρουσης μηδέν ώστε να υπολογιστούν εκ νέου τα μαθήματα που εμπλέκονται σε σύγκρουση μετά από κάθε επανάληψη. Η κωδικοποίηση είναι η εξής:

```
for (int i=0; i<num; i++) {
            Arhikos_pinakas[i].set_sugrousi(0);
}
```

3. Επίσης, μειώνεται η τιμή της count_mathimaton για τη μέρα εξέτασης του temp κατά ένα αφού ουσιαστικά αφαιρείται από αυτήν την μέρα. Η κωδικοποίηση είναι η εξής:

```
count_mathimaton[Arhikos_pinakas[temp].get_mera()]--;
```

Στη συνέχεια γίνεται ο έλεγχος των περιορισμών για να υπολογιστεί ο αριθμός των συγκρούσεων για την τρέχουσα επανάληψη (***sugrouseis1***). Δεν θα αναφερθούμε στο κομμάτι του κώδικα που αφορά τους περιορισμούς αφού έχει αναλυθεί σε προηγούμενο κεφάλαιο (κεφάλαιο 3, ενότητα 3.3)

4. Αφού υπολογιστεί ο αριθμός των συγκρούσεων εξετάζεται εάν ο αριθμός αυτός είναι μικρότερος από τις ελάχιστες συγκρούσεις (***min_sugrouseis***). Εάν ναι, αποθηκεύονται οι τιμές της μέρας,

ώρας και αίθουσας σε προσωρινές μεταβλητές και οι ελάχιστες συγκρούσεις παίρνουν την τιμή της μεταβλητής ***sugrouseis1***. Η κωδικοποίηση είναι η εξής:

```
if (sugrouseis1 <= min_sugrouseis) {
        temp_mera=i;
        temp_ora=j;
        temp_aithousa=k;
        min_sugrouseis=sugrouseis1;
 }
```

5. Στη συνέχεια μειώνεται η τιμή της count_mathimaton για τη μέρα εξέτασης i αφού αφαιρείται το μάθημα από εκεί για να του γίνει η νέα ανάθεση τιμών. Επίσης, ξαναγίνονται οι τιμές σύγκρουσης για όλα τα μαθήματα μηδέν ώστε να υπολογιστούν εκ νέου τα μαθήματα που εμπλέκονται σε σύγκρουση μετά από κάθε επανάληψη. Η κωδικοποίηση είναι η εξής:

```
count_mathimaton[i]--;
for (int g=0; g<num; g++) {
        Arhikos_pinakas[g].set_sugrousi(0);
}
```

6. Αφού έγιναν όλες οι επαναλήψεις αναθέτονται τελικά στο μάθημα temp οι τιμές που έχουν κρατηθεί στις προσωρινές μεταβλητές αφού θα περιέχουν τις τιμές για τις οποίες οι συγκρούσεις έχουν την μικρότερη δυνατή τιμή. Αυξάνεται επίσης η τιμή της count_mathimaton για τη μέρα εξέτασης i αφού εκεί θα τοποθετηθεί το μάθημα. Η κωδικοποίηση είναι η εξής:

```
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]++;
```

7. Τέλος, αυξάνεται η τιμή των max_epanalipsewn κατά ένα, επιστρέφεται ο αριθμός των min_sugrouseis και κλείνει η αρχική επανάληψη. Η κωδικοποίηση είναι η εξής:

```
max_epanalipseis++;
   return min_sugrouseis;
}
```

Αυτήν την διαδικασία εκτελεί ο αλγόριθμος μέχρι η τιμή των min_sugrouseis να γίνει 0 ή μέχρι η τιμή των max_epanalipseis να γίνει 50. Εάν η τιμή των max_epanalipseis ξεπεράσει το 50 γίνεται επανεκκίνηση του προγράμματος. Εφόσον το πρόγραμμα ξεπεράσει και το μέγιστο αριθμό επανεκκινήσεων χωρίς η τιμή min_sugrouseis να έχει γίνει 0, επιστρέφει αποτυχία.

4.3 Σύνοψη

Ο αλγόριθμος Min-conflicts with 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 έχουν conflicts.

ΚΕΦΑΛΑΙΟ 5

ΒΕΛΤΙΩΣΕΙΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

5.1 Εισαγωγή

Σκοπός της παρούσας διπλωματικής εργασίας είναι ο σχεδιασμός ενός ωρολογίου προγράμματος εξεταστικής περιόδου που να προσεγγίζει ρεαλιστικά το Πανεπιστήμιο Δυτικής Μακεδονίας και συγκεκριμένα το τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών στην Κοζάνη.

Το ωρολόγιο πρόγραμμα που έχει δημιουργηθεί με τη βοήθεια του αλγορίθμου Min-conflicts with random restarts και έχει αναλυθεί μέχρι την παρούσα φάση, μπορούμε να πούμε ότι καλύπτει σε ικανοποιητικό βαθμό τις βασικές ανάγκες του τμήματος. Αυτό όμως το γεγονός δεν συνεπάγει ότι δεν είναι δυνατόν το πρόγραμμα να δεχθεί κάποιες βελτιώσεις έτσι ώστε να το καταστήσουν ποιοτικά καλύτερο.

Στο κεφάλαιο αυτό λοιπόν θα παρουσιάσουμε ορισμένες βελτιώσεις στο πρόγραμμα όπως την προσθήκη ενός soft περιορισμού, μιας συνάρτησης βελτιστοποίησης f, καθώς και τη δημιουργία επιπλέον κώδικα με σκοπό την αναζήτηση pareto optimal λύσεων έτσι ώστε το πρόγραμμα να γίνει σαφώς πιο ρεαλιστικό και ποιοτικό.

5.2 Τεχνικές χειρισμού preferences

5.2.1 Περιορισμός Κόστους

Δημιουργήσαμε στο πρόγραμμα έναν εύκαμπτο περιορισμό οι οποίοι όπως έχουμε αναφέρει και σε προηγούμενο κεφάλαιο (κεφάλαιο 3) είναι απαραίτητοι για να γίνει ένα πρόγραμμα ρεαλιστικότερο και περισσότερο αποδοτικό.

Ο περιορισμός αυτός έχει να κάνει σχετικά με την προτίμηση του κάθε καθηγητή για τη μέρα που προτιμάει να εξεταστεί το μάθημά του. Αρχικά δημιουργήσαμε μια μεταβλητή με όνομα ***kostos*** για κάθε μάθημα. Η μεταβλητή αυτή θα είναι τύπου float. Αν η μέρα προτίμησης του καθηγητή για να διεξαχθεί το μάθημα (***Lista[i].get_protimisi()***) διαφέρει από τη μέρα όπου θα εξεταστεί το μάθημα

(***Arhikos_pinakas[i].get_mera()***) τότε η διαφορά των ημερών θα είναι η μεταβλητή ***kostos***. Στην περίπτωση όπου η μέρα προτίμησης από τον καθηγητή για να διεξαχθεί η εξέταση του μαθήματος συμπίπτει με τη μέρα που θα γίνει το μάθημα τότε δεν υπάρχει διαφορά στις μέρες και συνεπώς η μεταβλητή ***kostos*** θα πάρει την τιμή μηδέν. Η κωδικοποίηση του περιορισμού για το κόστος φαίνεται παρακάτω:

```
for (int i=0; i<num; i++) {
    if (Arhikos_pinakas[i].get_mera() == Lista[i].get_protimisi() ) {
                kostos[i]=0;
    }
    else if (Arhikos_pinakas[i].get_mera() > Lista[i].get_protimisi() ) {
                kostos[i]=Arhikos_pinakas[i].get_mera() - Lista[i].get_protimisi() ;
    }
    else if (Lista[i].get_protimisi() > Arhikos_pinakas[i].get_mera() ) {
               kostos[i]=(Lista[i].get_protimisi() - Arhikos_pinakas[i].get_mera() ;
    }
 }
```

Αφού λοιπόν βρήκαμε τις τιμές του κόστους για κάθε μάθημα, δημιουργήσαμε στη συνέχεια μια νέα μεταβλητή με όνομα ***sum_kostos*** και η οποία θα αθροίζει όλα τα κόστη. Να αναφέρουμε ότι η μεταβλητή είναι και αυτή τύπου float. Αφού έχουμε αρχικοποιήσει τη sum_kostos, προσθέτουμε σε αυτή το κόστος που έχουμε υπολογίσει από κάθε μάθημα. Ας δούμε τώρα την κωδικοποίηση:

```
sum_kostos = 0;
for (int p=0; p<num; p++) {
  sum_kostos = sum_kostos + kostos[p];
}
```

Με την προσθήκη του περιορισμού κόστους δημιουργήθηκε μια σημαντική διαφορά στον έλεγχο του αριθμού συγκρούσεων. Υπενθυμίζουμε ότι αν ο αριθμός των συγκρούσεων που βρέθηκαν στην τρέχουσα επανάληψη (***sugrouseis1***) είναι μικρότερος ή ίσος με τις ελάχιστες συγκρούσεις (***min_sugrouseis***) αποθηκεύονται οι

τιμές της μέρας, ώρας και αίθουσας σε προσωρινές μεταβλητές και οι ελάχιστες συγκρούσεις παίρνουν την τιμή της μεταβλητής ***sugrouseis1***. Η διαφορά που υπάρχει τώρα στην τεχνική του περιορισμού κόστους είναι η εξής. Μόνο όταν ο αριθμός των συγκρούσεων της τρέχουσας επανάληψης είναι μικρότερος από τον αριθμό των ελάχιστων συγκρούσεων αποθηκεύονται οι τιμές της μέρας, ώρας και αίθουσας σε προσωρινές μεταβλητές και οι ελάχιστες συγκρούσεις παίρνουν την τιμή της μεταβλητής ***sugrouseis1***. Αν σε μία επανάληψη προκύψουν πολλές ισοδύναμες καταστάσεις με βάση των αριθμό των συγκρούσεων, τότε επιλέγεται η καλύτερη με βάση το ολικό κόστος (***sum_kostos***). Αν το ολικό κόστος της τρέχουσας επανάληψης είναι μικρότερο από το ελάχιστο ολικό κόστος (***min_kostos***) αποθηκεύονται οι τιμές της μέρας, ώρας και αίθουσας σε προσωρινές μεταβλητές και η τιμή του ολικού κόστους ως ελάχιστο ολικό κόστος. Αποθηκεύεται ξανά όμως και η τιμή της μεταβλητής ***sugrouseis1*** ως ελάχιστες συγκρούσεις. Η κωδικοποίηση είναι η εξής:

```
if (sugrouseis1 < min_sugrouseis) {
            temp_mera=i;
            temp_ora=j;
            temp_aithousa=k;
            min_sugrouseis=sugrouseis1;
}
if (sugrouseis1 == min_sugrouseis) {
           temp_mera=i;
            temp_ora=j;
            temp_aithousa=k;
           min_sugrouseis=sugrouseis1;
     if (sum_kostos < min_kostos) {
            temp_mera=i;
            temp_ora=j;
            temp_aithousa=k;
           min_sugrouseis=sugrouseis1;
           min_kostos=sum_kostos;
     }
}
```

Η διαφορά που προκύπτει στον έλεγχο συγκρούσεων από την προσθήκη του περιορισμού κόστους είναι πολύ σημαντική γιατί όταν ο αριθμός των τρέχων συγκρούσεων με των ελάχιστων ήταν ίσος ουσιαστικά δεν μεταβάλλονταν κάτι ενώ τώρα όταν οι συγκρούσεις είναι ίσες ο αλγόριθμος προσπαθεί όσο το δυνατόν περισσότερο να μειώσει το ολικό κόστος. Αν ο αλγόριθμος καταφέρει παράλληλα με το μηδενισμό των συγκρούσεων να μηδενίσει και το ολικό κόστος τότε το τελικό ωρολόγιο πρόγραμμα εξεταστικής που θα εμφανίσει το πρόγραμμα αποτελεί τη βέλτιστη λύση.

5.2.2 Συνάρτηση Βελτιστοποίησης Fx

Δημιουργήσαμε στο πρόγραμμα μια συνάρτηση Fx η οποία θα περιλαμβάνει τόσο τον αριθμό συγκρούσεων όσο και τον αριθμό κόστους. Για να προκύψει βέλτιστη λύση χρειάζεται να μηδενιστεί η τιμή της συνάρτησης. Επειδή αυτά τα δύο είναι διαφορετικά πράγματα, θα χρειαστεί πρώτα να τα εκφράσω ως % ποσοστά και στη συνέχεια να τα προσθέσω έτσι ώστε να προκύψει μια τιμή της συνάρτησης Fx. Ας δούμε πως θα γίνει αυτό.

Αρχικά δημιούργησα τρεις νέες μεταβλητές:

1. Τη μεταβλητή ***pos_sugr*** όπου είναι το ποσοστό των συγκρούσεων. Η μεταβλητή αυτή θα είναι float. Επειδή έχουμε 7 πιθανούς περιορισμούς για μια σύγκρουση και επειδή το σύνολο των μαθημάτων είναι 41 θα έχουμε: 41*7=287 μέγιστες συγκρούσεις (***max_sugrouseis***) όπου θα μπει στον παρονομαστή και είναι σταθερός αριθμός. Στον αριθμητή θα έχουμε τη μεταβλητή ***sugrouseis1*** όπου υπολογίζει το συνολικό αριθμό συγκρούσεων. Παρακάτω παρουσιάζεται η εξίσωση και πως αυτή προκύπτει στον κώδικα:

```
pos_sugr = sugrouseis1*100/max_sugrouseis;
```

2. Τη μεταβλητή ***pos_kost*** όπου είναι το ποσοστό κόστους από τις προτιμήσεις των καθηγητών. Η μεταβλητή αυτή θα είναι float. Στον παρονομαστή θα έχουμε το συνολικό μέγιστο κόστος προτίμησης που δημιουργείται από κάθε μάθημα (***max_kost***).Αυτό προκύπτει ως εξής. Αν π.χ. η μέρα προτίμησης διεξαγωγής του μαθήματος ήταν η έκτη μέρα τότε ελέγχω αν η διαφορά 14-5 είναι μεγαλύτερη από το

5 και αναλόγως επιλέγω. Στο παράδειγμα εδώ θα επιλέξω το 14-5 και θα το προσθέσω στο ***max_kost***. Ο κώδικας είναι ο παρακάτω:

```
max_kost=0;
for (int i=0;i<num; i++){
   if (14 - Lista[i].get_protimisi() >= Lista[i].get_protimisi() ) {
            max_kost=max_kost + 14 - Lista[i].get_protimisi();
   }
   else  {
          max_kost=max_kost + Lista[i].get_protimisi();
   }
}
```

Στον αριθμητή θα έχω τη μεταβλητή ***sum_kostos*** όπου υπολογίζει το συνολικό κόστος από τις προτιμήσεις των καθηγητών. Έτσι, αφού δημιουργήθηκε η εξίσωση, η κωδικοποίηση είναι η εξής:

```
pos_kost = sum_kostos*100/max_kost;
```

3. Τη μεταβλητή Fx όπου είναι η συνάρτηση μας. Η Fx θα είναι και αυτή float αφού προκύπτει από τις δύο παραπάνω μεταβλητές που είναι και αυτές float. Η κωδικοποίηση της συνάρτησης θα είναι:

```
Fx = 0.85*arx_pos_sugr + 0.15 *arx_pos_kost;
```

Όπου τα 0.85 και 0.15 είναι τα βάρη του ποσοστού των συγκρούσεων και του ποσοστού κόστους αντίστοιχα. Η επιλογή των τιμών στα βάρη δεν είναι τυχαία. Έτρεξα το πρόγραμμα για κάθε περίπτωση πέντε φορές και έβγαλα το μέσο όρο. Έτσι προέκυψαν τα παρακάτω αποτελέσματα:

Βάρος Σύγκρουσης	**Βάρος Κόστους**	**Τιμή της Fx**	**Ποσοστό Συγκρούσεων (επί %)**	**Ποσοστό Κόστους (επί %)**
1	0	0,30	0,33	47,00
0,95	0,05	2,4	0,33	44,33
0,90	0,10	5,23	0,66	43,33
0,85	0,15	6,7	0,33	39,00
0,80	0,20	8,6	0,66	40,33
0,75	0,25	12,66	0,66	48,66
0,70	0,30	14,13	1,33	47,33
0,65	0,35	16,7	1,66	44,00
0,60	0,40	17,86	0,66	43,66
0,55	0,45	11,70	1,00	40,00

Πίνακας C: Αποτελέσματα για την επιλογή κατάλληλων βαρών για την Fx.

Όπως παρατηρούμε παραπάνω η καλύτερη περίπτωση είναι η τέταρτη όπου το βάρος για τις συγκρούσεις είναι 0,85 και το βάρος για το κόστος είναι 0,15 αντίστοιχα, αφού τα ποσοστά τους έχουν την μικρότερη δυνατή τιμή από αυτές του πίνακα. Σε άλλες δύο περιπτώσεις βέβαια το ποσοστό των συγκρούσεων είναι 0,33% αλλά εκεί το ποσοστό του κόστους είναι μεγαλύτερο από το 39% που συναντάμε στην τέταρτη περίπτωση γι' αυτό και απορρίπτονται.

Τέλος, με τη δημιουργία της συνάρτησης Fx ας δούμε τι γίνεται στον έλεγχο συγκρούσεων. Στην προηγούμενη περίπτωση με τον περιορισμό κόστους, ο αλγόριθμος αρχικά συνέκρινε εάν οι τρέχουσες συγκρούσεις ήταν λιγότερες από τις ελάχιστες ή στην περίπτωση όπου οι τρέχουσες συγκρούσεις ήταν ίσες με τις ελάχιστες εξέταζε το τρέχων κόστος με το ελάχιστο. Στην περίπτωσή μας εδώ, ο αλγόριθμος εξετάζει αποκλειστικά και μόνο την τιμή της τρέχουσας συνάρτησης με την ελάχιστη τιμή της συνάρτησης που έχει βρεθεί. Αν η τιμή της συνάρτησης (***Fx***) της τρέχουσας επανάληψης είναι μικρότερη ή ίση με την ελάχιστη τιμή της συνάρτησης (***min_Fx***) αποθηκεύονται οι τιμές της μέρας, ώρας και αίθουσας σε προσωρινές μεταβλητές και η ***min_Fx*** παίρνει την τιμή της ***Fx.*** Η κωδικοποίηση είναι η εξής:

```
if (Fx <= min_Fx) {
        temp_mera=i;
        temp_ora=j;
        temp_aithousa=k;
        min_Fx=Fx;
}
```

Τέλος, να αναφέρουμε ότι το τελικό ωρολόγιο πρόγραμμα εξεταστικής που θα εμφανίσει το πρόγραμμα αποτελεί τη βέλτιστη λύση στην περίπτωση όπου η Fx πάρει την τιμή μηδέν.

5.2.3 Pareto Optimality

Υπάρχουν προβλήματα στα οποία εμφανίζονται δύο ή περισσότερες αντικειμενικές συναρτήσεις και η λύση τους είναι ένα σύνολο λύσεων. Η αναζήτηση των βέλτιστων λύσεων και τα προβλήματα βελτιστοποίησης είναι γνωστά ως ***Πολυκριτηριακή Λήψη Αποφάσεων*** και ***Πολυαντικειμενικά Προβλήματα Βελτιστοποίησης*** (***ΠΠΒ***) αντίστοιχα. Ένα χαρακτηριστικό των ΠΠΒ είναι ότι δεν υπάρχει μία μοναδική βέλτιστη λύση αλλά ένα σύνολο μαθηματικά ισοδύναμων λύσεων που μπορούν να προσδιοριστούν. Αυτές οι λύσεις είναι γνωστές ως ***Pareto Optimal*** λύσεις και αποτελούν ακρότατες τιμές των αντικειμενικών συναρτήσεων.

Η δημιουργία του ωρολογίου προγράμματος εξεταστικής είναι ένα πολυαντικειμενικό πρόβλημα βελτιστοποίησης καθώς υπάρχει η συνάρτηση για τις συγκρούσεις αλλά και η συνάρτηση για τα κόστη. Σε τέτοιες περιπτώσεις λοιπόν δεν είναι σκόπιμη και ορθή η αναζήτηση του ακρότατου (μέγιστου ή ελάχιστου) για μια μόνο αντικειμενική συνάρτηση από τη στιγμή που και η άλλη που συντελεί στο πρόβλημα είναι εξίσου σημαντική. Έτσι γίνεται λόγος για εύρεση διαφορετικών λύσεων που μπορούν να δημιουργούν ένα είδος «εξισορρόπησης» μεταξύ των δύο διαφορετικών αντικειμενικών συναρτήσεων και είναι ανταγωνιστικές μεταξύ τους. Για παράδειγμα, μια λύση που μπορεί να είναι βέλτιστη ως προς τη μια αντικειμενική συνάρτηση ενδέχεται να μην ομοίως βέλτιστη για την άλλη. Επομένως, σε προβλήματα με περισσότερες της μιας ανταγωνιστικών αντικειμενικών συναρτήσεων, δεν υπάρχει μοναδική βέλτιστη λύση αλλά ένα (σύνολο) πλήθος λύσεων που είναι οι βέλτιστες.

Στα ΠΠΒ η βελτίωση που μπορεί να σημειωθεί σε μια εκ των αντικειμενικών συναρτήσεων μπορεί να σημάνει την αλλοίωση της άλλης. Κατά συνέπεια, η εύρεση λύσης δεν είναι τόσο άμεση όσο στα προβλήματα με μία αντικειμενική συνάρτηση βελτιστοποίησης. Δηλαδή η βελτίωση κάποιας εκ των αντικειμενικών συναρτησιακών τιμών δεν συνεπάγεται πάντοτε και βελτίωση της

άλλης που συντελεί στη λύση του προβλήματος. Στην περίπτωση αυτή κυριαρχεί η ιδέα του ***Pareto βέλτιστου*** (***Pareto Optimality***), όπου ένα σημείο είναι ***Pareto Optimal*** αν και μόνο αν δεν υπάρχει σημείο, τέτοιο ώστε η μετακίνηση από το σημείο αυτό να είναι εφικτή και ταυτόχρονα να σημειώνεται μείωση τουλάχιστον μιας αντικειμενικής συνάρτησης χωρίς να αυξάνεται κάποια άλλη.

5.2.3.1 Παράδειγμα Pareto Optimality

Ας δούμε τώρα ένα παράδειγμα για να κατανοήσουμε καλύτερα την έννοια του Pareto Optimality. Έστω τρία άτομα: η Άννα, ο Βασίλης και ο Γιάννης και τέσσερις πιθανές κατανομές ενός αγαθού: η Α, η Β, η Γ και η Δ. Ο παρακάτω πίνακας δείχνει πόσες μονάδες αυτού του αγαθού έχει ο καθένας σε κάθε κατανομή:

	Άννα	Βασίλης	Γιάννης
Α	1	1	1
Β	2	1	1
Γ	0	0	2
Δ	2	2	1

Θα συγκρίνουμε τις κατανομές αυτές με το κριτήριο του Pareto. Η κατανομή Β είναι *αποτελεσματικότερη* της Α κατά Pareto, διότι η Άννα είναι καλύτερα και ο Βασίλης και ο Γιάννης είναι στα ίδια. Η κατανομή όμως Β, δεν είναι και Pareto Optimal διότι υπάρχει και άλλη κατανομή, η Δ, η οποία είναι αποτελεσματικότερη κατά Pareto, εφόσον στην Δ η Άννα και ο Βασίλης έχουν από δύο και ο Γιάννης από 1. Παρατηρείστε ότι δεν υπάρχει άλλη κατανομή που μπορεί να βελτιώσει τουλάχιστον έναν από τους τρεις χωρίς να χειροτερεύσει κάποιον άλλον. Άρα η Δ είναι Pareto Optimal κατανομή. Επίσης και η κατάσταση Γ είναι Pareto Optimal. Αυτό συμβαίνει επειδή δεν υπάρχει άλλη κατανομή που να βελτιώνει τουλάχιστον έναν από τους τρεις χωρίς να χειροτερεύσει τους άλλους. Μπορεί η Δ να μοιράζει 5 μονάδες αγαθού (2+2+1) και η Γ μόνο 2, αλλά αυτό δεν μας ενδιαφέρει. Έτσι οι κατανομές Γ και Δ είναι οι κατανομές οι οποίες είναι Pareto Optimal.

5.2.3.2 Υλοποίηση Pareto Optimality

Ας δούμε τη διαδικασία που θα ακολουθήσουμε ώστε να εξάγουμε ***Pareto Optimal*** λύσεις στο πρόγραμμα μας. Δημιουργήσαμε μια μεταβλητή με όνομα ***pareto_optimal*** . Η μεταβλητή αυτή θα είναι int και αρχικά θα πάρει την τιμή 1. Η κωδικοποίηση είναι η εξής:

```
int pareto_optimal=1;
```

Στης συνέχεια, για κάθε μάθημα που εμπλέκεται σε σύγκρουση, γίνεται έλεγχος των περιορισμών για να υπολογιστεί ο αριθμός των συγκρούσεων για την τρέχουσα επανάληψη (***sugrouseis1***). Δεν θα αναφερθούμε στο κομμάτι του κώδικα που αφορά τους περιορισμούς αφού έχει αναλυθεί σε προηγούμενο κεφάλαιο (κεφάλαιο 3, ενότητα 3.3) Για να εισέλθει όμως στους περιορισμούς πρέπει η μεταβλητή ***pareto_optimal*** να είναι διάφορη του 0. Η κωδικοποίηση είναι η εξής:

```
for (int s=0;s<num ;s++) {
   if (Arhikos_pinakas[s].get_sugrousi()==1) {
                        for (int i=0;i<15;i++) {
                            for (int j=0;j<4;j++) {
                                    for (int k=0;k<5;k++) {
                                        if (pareto_optimal !=0) {
```

Αφού βρέθηκε ο αριθμός των τρέχων συγκρούσεων (***sugrouseis1***) και του τρέχοντος κόστους (***sum_kostos***) ελέγχουμε αν οι τρέχουσες συγκρούσεις είναι μεγαλύτερες ή ίσες από τις ελάχιστες και ταυτόχρονα αν το τρέχον κόστος είναι μεγαλύτερο ή ίσο από το ελάχιστο. Αν ισχύουν και τα δύο κριτήρια ταυτόχρονα τότε η μεταβλητή ***pareto_optimal*** παίρνει την τιμή 0. Η διαδικασία αυτή επαναλαμβάνεται μέχρι να τελειώσουν οι επαναλήψεις (τα τρία for που βρίσκονται παραπάνω). Η κωδικοποίηση είναι η εξής:

```
if ((sugrouseis1>=min_sugrouseis) && (sum_kostos>=min_kostos)) {
                        pareto_optimal=0;
}
```

Τελειώνοντας, εάν η μεταβλητή ***pareto_optimal*** έχει την τιμή 1 τότε είναι pareto_optimal λύση διαφορετικά αν έχει την τιμή 0 δεν είναι pareto_optimal. Όσες λύσεις είναι pareto_optimal αποθηκεύονται και εμφανίζονται σε ένα .txt αρχείο (pareto_optimal.txt).

5.3 Σύγκριση τεχνικών

Στο κεφάλαιο αυτό αναφέραμε και υλοποιήσαμε στο πρόγραμμα τρεις διαφορετικές τεχνικές χειρισμού των preferences. Στις παρακάτω ενότητες θα παρουσιάσουμε συνοπτικά τα πειραματικά αποτελέσματα από τις τρεις διαφορετικές τεχνικές σε ένα μέτριο πρόβλημα όσο και σε ένα δύσκολο καθώς και σε μερικές παραλλαγές αυτών των δύο κατηγοριών.

5.3.1 Μέτριο πρόβλημα

Σε αυτό το στάδιο της πειραματικής μελέτης 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 μέτριο πρόβλημα. Για το λόγο αυτό, θα υπάρχουν δέκα μαθήματα σε κάθε εξάμηνο έτσι ώστε να έχουμε συνολικά πενήντα μαθήματα. Πριν παρουσιάσουμε όμως τον πίνακα με τα πειραματικά αποτελέσματα είναι απαραίτητη η αναφορά μιας επισήμανσης.

Με τους περιορισμούς που έχουμε θέσει στον κώδικα το πρόγραμμα δεν είναι δυνατόν να βρει λύση ποτέ, εξ' αιτίας του περιορισμού όπου μαθήματα του ίδιου εξαμήνου δεν εξετάζονται σε διαδοχικές μέρες. Όπως γίνεται εύκολα κατανοητό με αυτόν τον περιορισμό στον κώδικα και σε συνδυασμό με την ύπαρξη δέκα μαθημάτων ανά εξάμηνο θα χρειαζόμασταν παραπάνω από τρεις βδομάδες εξεταστικής περιόδου (αφού σε μια εβδομάδα μπορούν να εξεταστούν το πολύ τρία μαθήματα του ίδιου εξαμήνου) κάτι το οποίο δεν είναι εφικτό. Έτσι οδηγούμαστε στην αφαίρεση του περιορισμού αυτού αφήνοντας τον κώδικα του υπόλοιπου προγράμματος ως έχει. Τα πειραματικά αποτελέσματα φαίνονται στον παρακάτω πίνακα:

	Περιορισμός Κόστους	Συνάρτηση Βελτιστοποίησης Fx	Pareto Optimality
Χρόνος Υλοποίησης	0,45 s	5,48 s	0,75 s
Συγκρούσεις	0	1,33	0
Κόστος	4,5	188,66	2,96

Πίνακας D: Πειραματικά Αποτελέσματα Τεχνικών Χειρισμών Preferences για μέτριο πρόβλημα.

Χρησιμοποιώντας την τεχνική Pareto Optimality το πρόγραμμα εξάγει σαν λύση αποτέλεσμα με μικρότερο αριθμό κόστους και συγκρούσεων συγκριτικά με τις άλλες δύο τεχνικές αλλά χρειάζεται ελάχιστα περισσότερο χρόνο ώστε να βρεθεί αυτή η λύση με αυτήν την τεχνική σε σχέση με την τεχνική του Περιορισμού Κόστους. Επίσης, ο αριθμός των συγκρούσεων για τις τεχνικές περιορισμού κόστους και Pareto Optimality είναι στο μηδέν πράγμα που σημαίνει ότι το πρόγραμμα βρίσκει πάντα λύση γι' αυτές τις δύο τεχνικές ενώ για την Συνάρτηση Βελτιστοποίησης Fx το πρόγραμμα εμφανίζει σχεδόν πάντα. Εάν χρησιμοποιήσουμε την τεχνική της Συνάρτησης Βελτιστοποίησης Fx το πρόγραμμα θα χρειαστεί τον περισσότερο χρόνο ώστε να βρει λύση και από τις τρεις τεχνικές καθώς επίσης εξάγει και τον μεγαλύτερο αριθμό κόστους.

Στο επόμενο βήμα της πειραματικής μελέτης, κρατάμε σταθερά όλα τα δεδομένα εισόδου που παίρνουμε από το αρχείο test.txt εκτός των προτιμήσεων των καθηγητών για το ποια μέρα θέλουν να εξεταστεί το μάθημα που διδάσκουν. Μεταβάλλοντας την μεταβλητή ***protimisi*** ας δούμε κατά πόσο μεταβάλλεται και η συμπεριφορά του αλγορίθμου. Τα πειραματικά αποτελέσματα φαίνονται στον παρακάτω πίνακα:

	Περιορισμός Κόστους	Συνάρτηση Βελτιστοποίησης Fx	Pareto Optimality
Χρόνος Υλοποίησης	0,41 s	4,22 s	0,86 s
Συγκρούσεις	0	0	0
Κόστος	3,83	164	4,66

Πίνακας E: Πειραματικά Αποτελέσματα Τεχνικών Χειρισμών Preferences για μέτριο πρόβλημα.

Με την αλλαγή της μεταβλητής ***protimisi*** παρατηρούμε ότι χρησιμοποιώντας την τεχνική του περιορισμού κόστους το πρόγραμμα εξάγει σαν λύση αποτέλεσμα με μικρότερο αριθμό κόστους συγκριτικά με το κόστος στην τεχνική Pareto Optimality, καθώς επίσης και σε μικρότερο χρονικό διάστημα σε σχέση με τις άλλες δύο τεχνικές. Η τεχνική της Συνάρτησης Βελτιστοποίησης Fx εξάγει πάλι με το μεγαλύτερο κόστος, αν και εμφανίζεται βελτιωμένη σε σχέση με τον πίνακα D. Τέλος, αυτή τη φορά το πρόγραμμα εμφανίζει πάντα λύση και για τρεις τεχνικές χειρισμού των preferences αφού ο αριθμός των συγκρούσεων είναι μηδέν, που σημαίνει ότι με τις τυχαίες αλλαγές στις προτιμήσεις των καθηγητών για το ποια μέρα προτιμούν να εξεταστεί το μάθημά τους το πρόγραμμα εμφανίζει με μεγαλύτερη ευκολία λύση απ' ότι προηγουμένως στον πίνακα D.

5.3.2 Δύσκολο πρόβλημα

Στην κατηγορία αυτή ο αριθμός των μαθημάτων ανά εξάμηνο κυμαίνεται από 6 έως 7 μαθήματα αλλά αυξάνεται ο αριθμός των περιορισμών σε σχέση με το μέτριο πρόβλημα που αναλύσαμε παραπάνω γι' αυτό και θεωρείται δύσκολο αυτό το πρόβλημα. Τα πειραματικά αποτελέσματα φαίνονται στον παρακάτω πίνακα:

	Περιορισμός Κόστους	Συνάρτηση Βελτιστοποίησης Fx	Pareto Optimality
Χρόνος Υλοποίησης	1,1 s	4,02 s	3,16 s
Συγκρούσεις	0,33	0,33	0,33
Κόστος	7,33	123,66	4,27

Πίνακας F: Πειραματικά Αποτελέσματα Τεχνικών Χειρισμών Preferences για δύσκολο πρόβλημα.

Εξ αιτίας της αύξησης των περιορισμών και συνεπώς της αύξησης της πολυπλοκότητας παρατηρούμε μια γενική αύξηση στις τιμές του χρόνου υλοποίησης, των συγκρούσεων και του κόστους και για τις τρεις τεχνικές χειρισμού των preferences. Μπορούμε να παρατηρήσουμε ότι χρησιμοποιώντας την τεχνική Pareto Optimality το πρόγραμμα εξάγει σαν λύση αποτέλεσμα με μικρότερο αριθμό κόστους συγκριτικά με τις άλλες δύο τεχνικές αλλά χρειάζεται περισσότερο χρόνο ώστε να βρεθεί αυτή η λύση με αυτήν την τεχνική σε σχέση με την τεχνική του Περιορισμού

Κόστους. Επίσης, ο αριθμός των συγκρούσεων τείνει στο μηδέν πράγμα που σημαίνει ότι το πρόγραμμα βρίσκει σχεδόν πάντα λύση ανεξάρτητα με την τεχνική που χρησιμοποιούμε. Εάν χρησιμοποιήσουμε την τεχνική της Συνάρτησης Βελτιστοποίησης Fx το πρόγραμμα θα χρειαστεί τον περισσότερο χρόνο ώστε να βρει λύση και από τις τρεις τεχνικές καθώς επίσης εξάγει και τον μεγαλύτερο αριθμό κόστους.

Στην τελευταία πειραματική μελέτη που θα πραγματοποιηθεί, κρατάμε σταθερά όλα τα δεδομένα εισόδου που παίρνουμε από το αρχείο test.txt εκτός των προτιμήσεων των καθηγητών για το ποια μέρα θέλουν να εξεταστεί το μάθημα που διδάσκουν. Μεταβάλλοντας την μεταβλητή ***protimisi*** ας δούμε κατά πόσο μεταβάλλεται και η συμπεριφορά του αλγορίθμου. Τα πειραματικά αποτελέσματα φαίνονται στον παρακάτω πίνακα:

	Περιορισμός Κόστους	Συνάρτηση Βελτιστοποίησης Fx	Pareto Optimality
Χρόνος Υλοποίησης	2,12 s	3,35 s	4,23 s
Συγκρούσεις	0,16	0,16	0,16
Κόστος	5,16	116,83	5,29

Πίνακας G: Πειραματικά Αποτελέσματα Τεχνικών Χειρισμών Preferences για δύσκολο πρόβλημα.

Με την αλλαγή της μεταβλητής ***protimisi*** παρατηρούμε ότι χρησιμοποιώντας την τεχνική του περιορισμού κόστους το πρόγραμμα έστω και οριακά εξάγει σαν λύση αποτέλεσμα με μικρότερο αριθμό κόστους συγκριτικά με το κόστος στην τεχνική Pareto Optimality, καθώς επίσης και σε μικρότερο χρονικό διάστημα σε σχέση με τις άλλες δύο τεχνικές. Η τεχνική της Συνάρτησης Βελτιστοποίησης Fx εξάγει πάλι με το μεγαλύτερο κόστος, αν και εμφανίζεται βελτιωμένη σε σχέση με τον πίνακα F. Τέλος, ο αριθμός των συγκρούσεων, αν και τείνει εκ νέου στο μηδέν και για τις τρεις τεχνικές (το πρόγραμμα δηλαδή βρίσκει σχεδόν πάντα λύση), είναι μικρότερος συγκριτικά με τον αριθμό των συγκρούσεων στον πίνακα F που σημαίνει ότι με τις τυχαίες αλλαγές στις προτιμήσεις των καθηγητών για το ποια μέρα προτιμούν να εξεταστεί το μάθημά τους το πρόγραμμα εμφανίζει ευκολότερα λύση απ’ ότι προηγουμένως.

ΚΕΦΑΛΑΙΟ 6

ΣΥΜΠΕΡΑΣΜΑΤΑ

Στην παρούσα διπλωματική εργασία αναπτύχθηκε ένα σύστημα κατάρτισης ωρολογίου προγράμματος εξεταστικής περιόδου για το Πανεπιστήμιο Δυτικής Μακεδονίας και συγκεκριμένα για το τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών. Ο σχεδιασμός του μοντέλου πραγματοποιήθηκε με βάση τη θεωρία των προβλημάτων ικανοποίησης περιορισμών, δίνοντας έτσι στον χρήστη τη δυνατότητα να εκφράσει τους περιορισμούς που ισχύουν στο πρόβλημα. Η αυτόματη κατάρτιση ωρολογίου προγράμματος αποτελεί ένα πολύ δύσκολο πρόβλημα καθώς ανήκει στην κατηγορία των NP-complete προβλημάτων όπου ο χρόνος αναζήτησης λύσης αυξάνει εκθετικά σε σχέση με τον αριθμό των δεδομένων.

Ο αλγόριθμος που χρησιμοποιήθηκε είναι ο Min-conflicts with random restarts και ανήκει στην κατηγορία των αλγορίθμων τοπικής αναζήτησης οι οποίοι έχουν πολύ καλή απόδοση σε προβλήματα όπου δεν έχουν μεγάλο μέγεθος. Για να γίνει το πρόγραμμα περισσότερο ρεαλιστικό και αποδοτικό χρησιμοποιήθηκαν τρεις διαφορετικές τεχνικές χειρισμού των προτιμήσεων (ο περιορισμός κόστους, η συνάρτηση βελτιστοποίησης Fx και το Pareto Optimality) χωρίς όμως να γίνει σύγκριση των λύσεων που εμφανίζει κάθε τεχνική, για να εξετάσουμε ποια είναι η καλύτερη, μιας και το πρόβλημά μας είναι Πολυκριτηριακό Πρόβλημα Βελτιστοποίησης και όλες οι λύσεις είναι εξίσου σημαντικές. Μόνο ο χρήστης μπορεί να κρίνει ποια τεχνική τον ικανοποιεί περισσότερο αναλόγως με τις προτιμήσεις που υπάρχουν στο εκάστοτε πρόβλημα.

Παρόλο που οι βελτιώσεις που έγιναν ήταν αρκετές, το πρόγραμμα ωστόσο μπορεί να δεχθεί μια ακόμα σημαντική βελτίωση. Θα μπορούσε το περιβάλλον της εφαρμογής να είναι *δυναμικό*, δηλαδή να μπορεί το πρόγραμμα να μεταβάλλεται αν συμβεί κάτι απρόοπτο. Το απρόοπτο που ενδεχομένων να συμβεί στην περίπτωση του ακαδημαϊκού προγράμματος είναι να μην έρθει ένας καθηγητής λόγω υγείας ή να γίνει κατάληψη, με αποτέλεσμα να μην πραγματοποιηθεί η εξέταση του μαθήματος την προγραμματισμένη ημερομηνία διεξαγωγής του. Η δυσκολία της υλοποίησης

σχετίζεται με την έγκαιρη ενημέρωση της εφαρμογής και όσων εμπλέκονται έμμεσα σε αυτήν ,δηλαδή των φοιτητών και των καθηγητών.

Μία τελευταία σημαντική βελτίωση θα ήταν η δημιουργία ενός script. Όταν το πρόγραμμα εμφάνιζε βέλτιστη λύση, δηλαδή λύση με μηδέν αριθμό συγκρούσεων και κόστους, τότε αυτή η λύση θα ανεβαίνει ως πρόγραμμα εξεταστικής αυτόματα στο site της σχολής με τη μορφή ανακοίνωσης.

ΒΙΒΛΙΟΓΡΑΦΙΑ

1) "Scheduling, Timetabling and Rostering – A special relationship?" A. Wren, 1996.
2) A. Schaeft, "A Survey of Automated Timetabling", TR CS-R9567, CWI-Cent. Wiskunde en Informatica, 1995.
3) 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.
4) "A survey of practical applications of examination timetabling problems", Carter M.W, 1986.
5) "A Language for Specifying Complete Timetabling Problems", L.Reis, E.Oliveira.
6) "Nearly Optimum Timetable Construction Through CLP and Intelligent Search", Panagiotis Stamatopoulos, Efstratios Viglas and Serafeim Karaboyas, 1998.
7) "UTSE: Construction of Optimum Timetables for University Courses – A CLP Based Approach", Harikleia Frangouli, Vassilis Harmadas and Panagiotis Stamatopoulos, 1995.
8) "Timetabling for Greek high schools", Birbas T., Daskalaki S., & Housos E. 1997.
9) "Recent research directions in automated timetabling", Springer -Verlag. Burke E.K., & Petrovic S. 2002.
10) "Constraint Satisfaction with Preferences", Hana Rudova, January 2001.
11) "Constraint Programming: In Pursuit of the Holy Grail", Roman Bartak, 1998.
12) "Αλγόριθμοι Συνδυαστικής Βελτιστοποίησης με Έμφαση σε Μεταευρετικές Τεχνικές", ΧΡΗΣΤΟΣ Γ. ΓΚΟΓΚΟΣ, 2009.
13) "Artificial Intelligence, A Modern Approach", Stuart Russel and Peter Norvig, 2003.
14) "Constraint –Based Timetabling", Tomas Müller, 2005.
15) M.Carey & D.Johnson, "Computers and Intractability: a guide to the theory of NP-Completeness", Freeman 1979.
16) "Timetabling university examinations", Johnson D, 1990.
17) J. Pearl. "Heuristics: Intelligent search strategies for computer problem solving." Addison -Wesley, Reading, MA, 1984.
18) D. De Werra. "An introduction to timetabling". European Journal of Operational Research.
19) "Constraint Satisfaction", R.Dechter, University of California, and F.Rossi, University of Padova, 1998.
20) "Βελτιστοποίηση Χρονοπρογραμματισμού με Χρήση Γενετικών Αλγορίθμων", Αλέξανδρος Μ. Κελεμένης, 2003.

21) “Pareto Optimality”, Georgia Institute of Technology Systems Realization Laboratory.
22) “Η έννοια της αποτελεσματικότητας κατά Pareto”,Νίκος Θεοχαράκης, 2009.
23) “Βέλτιστος σχεδιασμός κατασκευών με πολλαπλά κριτήρια με χρήση στρατηγικών εξέλιξης”, Ευάγγελου Ε. Πλεύρη, 2001.

ΠΑΡΑΡΤΗΜΑ Α

ΠΕΡΙΟΡΙΣΜΟΣ ΚΟΣΤΟΥΣ

```
#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 kodikos_kath;
		int eks;
		int protimisi;
		int arithmos_foititwn;
public:
		void set_onoma_math(std::string string1){onoma_math = string1;}
		std::string get_onoma_math(){return onoma_math;}
		void set_onoma_kath(std::string string){onoma_kath = string;}
		std::string get_onoma_kath(){return onoma_kath;}
		void set_kodiko_kath(int k){kodikos_kath = k;}
		int get_kodiko_kath(){return kodikos_kath;}
		void set_eks(int e){eks = e;}
		int get_eks(){return eks;}
		void set_protimisi (int pr) {protimisi = pr; }
		int get_protimisi() {return protimisi; }
		void set_arithmos_foititwn(int ar_foit){arithmos_foititwn = ar_foit;}
		int get_arithmos_foititwn(){return arithmos_foititwn;}
};

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 print_programma();
};
```

```
void Eksetastiki::create_lista(){
		std::string onoma_math;
		std::string onoma_kath;
		int kodikos_kath;
		int eks;
		int protimisi;
		int arithmos_foititwn;
		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 >> kodikos_kath >> eks >> protimisi >> arithmos_foititwn;
			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 >> kodikos_kath >> eks >> protimisi >> arithmos_foititwn;
			Lista[i].set_onoma_math(onoma_math);
			Lista[i].set_onoma_kath(onoma_kath);
			Lista[i].set_kodiko_kath(kodikos_kath);
			Lista[i].set_eks(eks);
			Lista[i].set_protimisi(protimisi);
			Lista[i].set_arithmos_foititwn(arithmos_foititwn);
			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() << " protimaei na eksetastei tin imera: " <<
Lista[i].get_protimisi() << " kai tha eksetastoun " << Lista[i].get_arithmos_foititwn() << " foitites." << endl;
	}
}

void Eksetastiki::create_arhiko_pinaka() {
 int oliko_kostos=0;
 int kostos[num];
 for (int i=0; i<num; i++) {
     kostos[i]=0;
 }
		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()%5);
			Arhikos_pinakas[i].set_aithousa(temp);
			Arhikos_pinakas[i].set_sugrousi(0);
		}
 for (int i=0; i<num; i++) {
    if (Arhikos_pinakas[i].get_mera() == Lista[i].get_protimisi() ) {
		kostos[i]=0;
    }
  else if (Arhikos_pinakas[i].get_mera() > Lista[i].get_protimisi() ) {
		kostos[i]=Arhikos_pinakas[i].get_mera() - Lista[i].get_protimisi() ;
    }
```

```
 else if (Lista[i].get_protimisi() > Arhikos_pinakas[i].get_mera() ) {
                kostos[i]=(Lista[i].get_protimisi() - Arhikos_pinakas[i].get_mera() );
    }
 }
  for (int i=0; i<num; i++) {
    oliko_kostos = oliko_kostos + kostos[i];
   }
}

int Eksetastiki::eleghos_sugrouseon() {
               int count_sugkrousewn=0;
               int count_mathimaton[15];
              int xwr_aithousas[5];
              xwr_aithousas[0] = 40;
               xwr_aithousas[1] = 60;
               xwr_aithousas[2] = 100;
               xwr_aithousas[3] = 150;
               xwr_aithousas[4] = 250;
              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;
               }
 int kostos[num];
 for (int i=0; i<num; i++) {
 kostos[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_kodiko_kath() == Lista[j].get_kodiko_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++;
                                       }
```

```
                                        }
                                }
                        }
for (int i=0;i<num; i++) {
      if (Lista[i].get_arithmos_foititwn() > xwr_aithousas[Arhikos_pinakas[i].get_aithousa()]) {
         Arhikos_pinakas[i].set_sugrousi(1);
         count_sugkrousewn++;
      }
}
int max_epanalipseis=0;
int epoxes1=0;
int min_sugrouseis=count_sugkrousewn;
int oliko_kostos;
int min_kostos=oliko_kostos;
int sum_kostos=0;
int sugrouseis1;
int temp_mera;
int temp_ora;
int temp_aithousa;
int temp_sugrouseis[num];
while ((max_epanalipseis<50) && (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<5;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 s=0;s<num; s++) {
     if (Lista[s].get_arithmos_foititwn() > xwr_aithousas[Arhikos_pinakas[s].get_aithousa()]) {
                    Arhikos_pinakas[s].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_kodiko_kath() == Lista[n].get_kodiko_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);
                       }
              }
          }
    }
}
for (int p=0; p<num; p++) {
                if (Arhikos_pinakas[p].get_mera() == Lista[p].get_protimisi() ) {
                      kostos[p]=0;
                }
                else if (Arhikos_pinakas[p].get_mera() > Lista[p].get_protimisi() ) {
                    kostos[p]=Arhikos_pinakas[p].get_mera() - Lista[p].get_protimisi() ;
                }
                else if (Lista[p].get_protimisi() > Arhikos_pinakas[p].get_mera() ) {
                       kostos[p]=(Lista[p].get_protimisi() - Arhikos_pinakas[p].get_mera() );
                }
}
sum_kostos = 0;
for (int p=0; p<num; p++) {
    sum_kostos = sum_kostos + kostos[p];
}
     if (sugrouseis1 < min_sugrouseis) {
                temp_mera=i;
                temp_ora=j;
                temp_aithousa=k;
                min_sugrouseis=sugrouseis1;
      }
     if (sugrouseis1 == min_sugrouseis) {
                      temp_mera=i;
                       temp_ora=j;
                       temp_aithousa=k;
                      min_sugrouseis=sugrouseis1;
          if (sum_kostos < min_kostos) {
                     temp_mera=i;
                     temp_ora=j;
                     temp_aithousa=k;
                     min_sugrouseis=sugrouseis1;
                     min_kostos=sum_kostos;
           }
      }
      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++;
```

```
}
cout << "max_epanalipseis: " << max_epanalipseis << endl;
cout << "sugrouseis: " << min_sugrouseis << endl;
cout << "To sinoliko kostos apo tis protimiseis twn kathigitwn einai : " << min_kostos << endl;
return min_sugrouseis;
}

void Eksetastiki::print_programma() {

    ofstream fp;
    fp.open("programma eksetastikis.doc");
                        fp << "DATE                                    |          AITHOUSA |          MATHIMA"<< endl;
                        fp << "_____________________________________________________________________________ "<< endl;



    for(int z=0;z<num; z++){
     if (Arhikos_pinakas[z].get_ora()==0) {
       if (Arhikos_pinakas[z].get_aithousa()==0) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (9  - 12)       |          " << "A                |          " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(9  - 12)       |          " << "A                |          " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==1) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (9  - 12)       |          " << "B                |          " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(9  - 12)       |          " << "B                |          " <<
Lista[z].get_onoma_math() << endl;

       }
        else if (Arhikos_pinakas[z].get_aithousa()==2) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (9  - 12)       |          " << "C                |          " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(9  - 12)       |          " << "C                |          " <<
Lista[z].get_onoma_math() << endl;

       }
        else if (Arhikos_pinakas[z].get_aithousa()==3) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (9  - 12)       |          " << "D                |          " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(9  - 12)       |          " << "D                |          " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==4) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (9  - 12)       |          " << "E                |          " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(9  - 12)       |          " << "E                |          " <<
Lista[z].get_onoma_math() << endl;

        }
      }
      else if (Arhikos_pinakas[z].get_ora()==1) {
        if (Arhikos_pinakas[z].get_aithousa()==0) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (12 - 15)       |          " << "A                |          " <<
Lista[z].get_onoma_math() << endl;
            else
```

```
                        fp << Arhikos_pinakas[z].get_mera() << ",(12 - 15)      |       " << "A             |       " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==1) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (12 - 15)     |       " << "B             |       " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(12 - 15)      |       " << "B             |       " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==2) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (12 - 15)     |       " << "C             |       " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(12 - 15)      |       " << "C             |       " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==3) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (12 - 15)     |       " << "D             |       " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(12 - 15)      |       " << "D             |       " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==4) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (12 - 15)     |       " << "E             |       " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(12 - 15)      |       " << "E             |       " <<
Lista[z].get_onoma_math() << endl;

        }
      }
      else if (Arhikos_pinakas[z].get_ora()==2) {
        if (Arhikos_pinakas[z].get_aithousa()==0) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (15 - 18)     |       " << "A             |       " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(15 - 18)      |       " << "A             |       " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==1) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (15 - 18)     |       " << "B             |       " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(15 - 18)      |       " << "B             |       " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==2) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (15 - 18)     |       " << "C             |       " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(15 - 18)      |       " << "C             |       " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==3) {
            if(Arhikos_pinakas[z].get_mera()<10)
```

```
                        fp << Arhikos_pinakas[z].get_mera() << ", (15 - 18)       |          " << "D                |          " <<
Lista[z].get_onoma_math() << endl;
                else
                        fp << Arhikos_pinakas[z].get_mera() << ",(15 - 18)       |          " << "D                |          " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==4) {
                if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (15 - 18)       |          " << "E                |          " <<
Lista[z].get_onoma_math() << endl;
                else
                        fp << Arhikos_pinakas[z].get_mera() << ",(15 - 18)       |          " << "E                |          " <<
Lista[z].get_onoma_math() << endl;

        }
      }
      else if (Arhikos_pinakas[z].get_ora()==3) {
        if (Arhikos_pinakas[z].get_aithousa()==0) {
                if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (18 - 21)       |          " << "A                |          " <<
Lista[z].get_onoma_math() << endl;
                else
                        fp << Arhikos_pinakas[z].get_mera() << ",(18 - 21)       |          " << "A                |          " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==1) {
                if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (18 - 21)       |          " << "B                |          " <<
Lista[z].get_onoma_math() << endl;
                else
                        fp << Arhikos_pinakas[z].get_mera() << ",(18 - 21)       |          " << "B                |          " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==2) {
                if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (18 - 21)       |          " << "C                |          " <<
Lista[z].get_onoma_math() << endl;
                else
                        fp << Arhikos_pinakas[z].get_mera() << ",(18 - 21)       |          " << "C                |          " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==3) {
                if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (18 - 21)       |          " << "D                |          " <<
Lista[z].get_onoma_math() << endl;
                else
                        fp << Arhikos_pinakas[z].get_mera() << ",(18 - 21)       |          " << "D                |          " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==4) {
                if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (18 - 21)       |          " << "E                |          " <<
Lista[z].get_onoma_math() << endl;
                else
                        fp << Arhikos_pinakas[z].get_mera() << ",(18 - 21)       |          " << "E                |          " <<
Lista[z].get_onoma_math() << endl;

        }
      }
                }
    fp.close();
}

void Eksetastiki::print_teliko_pinaka() {

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

```
   if (Arhikos_pinakas[z].get_ora()==0) {
        if (Arhikos_pinakas[z].get_aithousa()==0) {
          cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 9 ews 12"  << ",stin aithousa A." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==1) {
           cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 9 ews 12"  << ",stin aithousa B." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==2) {
           cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 9 ews 12"  << ",stin aithousa C." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==3) {
           cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 9 ews 12"  << ",stin aithousa D." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==4) {
          cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 9 ews 12"  << ",stin aithousa E." << endl;
        }
    }
    else if (Arhikos_pinakas[z].get_ora()==1) {
        if (Arhikos_pinakas[z].get_aithousa()==0) {
           cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 12 ews 3"  << ",stin aithousa A." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==1) {
           cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 12 ews 3"  << ",stin aithousa B." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==2) {
           cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 12 ews 3"  << ",stin aithousa C." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==3) {
           cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 12 ews 3"  << ",stin aithousa D." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==4) {
           cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 12 ews 3"  << ",stin aithousa E." << endl;
        }
    }
   else if (Arhikos_pinakas[z].get_ora()==2) {
        if (Arhikos_pinakas[z].get_aithousa()==0) {
          cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 3 ews 6"  << ",stin aithousa A." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==1) {
           cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 3 ews 6"  << ",stin aithousa B." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==2) {
           cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 3 ews 6"  << ",stin aithousa C." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==3) {
           cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 3 ews 6"  << ",stin aithousa D." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==4) {
          cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 3 ews 6"  << ",stin aithousa E." << endl;
        }
     }
    else if (Arhikos_pinakas[z].get_ora()==3) {
        if (Arhikos_pinakas[z].get_aithousa()==0) {
           cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 6 ews 9"  << ",stin aithousa A." << endl;
        }
```

```
        else if (Arhikos_pinakas[z].get_aithousa()==1) {
           cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 6 ews 9"  << ",stin aithousa B." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==2) {
           cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 6 ews 9"  << ",stin aithousa C." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==3) {
           cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 6 ews 9"  << ",stin aithousa D." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==4) {
          cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 6 ews 9"  << ",stin aithousa E." << 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 < 50 );
              cout << "epanekkiniseis: " << max_epanekkiniseis-1 << endl;
              if (min_sygrouseis ==0){
                    Ex.print_programma();
                    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;
}
```

ΠΑΡΑΡΤΗΜΑ Β

ΣΥΝΑΡΤΗΣΗ ΒΕΛΤΙΣΤΟΠΟΙΗΣΗΣ Fx

```
#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 kodikos_kath;
		int eks;
		int protimisi;
		int arithmos_foititwn;
public:
		void set_onoma_math(std::string string1){onoma_math = string1;}
		std::string get_onoma_math(){return onoma_math;}
		void set_onoma_kath(std::string string){onoma_kath = string;}
		std::string get_onoma_kath(){return onoma_kath;}
		void set_kodiko_kath(int k){kodikos_kath = k;}
		int get_kodiko_kath(){return kodikos_kath;}
		void set_eks(int e){eks = e;}
		int get_eks(){return eks;}
		void set_protimisi (int pr) {protimisi=pr; }
		int get_protimisi() {return protimisi; }
		void set_arithmos_foititwn(int ar_foit){arithmos_foititwn = ar_foit;}
		int get_arithmos_foititwn(){return arithmos_foititwn;}
};

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 print_programma();
};
```

```
void Eksetastiki::create_lista(){
		std::string onoma_math;
		std::string onoma_kath;
		int kodikos_kath;
		int eks;
		int protimisi;
		int count=0;
		int arithmos_foititwn;
		ifstream eisodos1,eisodos2;
		eisodos1.open("test.txt");
		if(eisodos1 == NULL) {
			cout << "Apotuxia anoigmatos arxeiou. Eksodos" << endl;
		}
		while(!eisodos1.eof()) {
			eisodos1 >> onoma_math >> onoma_kath >> kodikos_kath >> eks >> protimisi >> arithmos_foititwn;
			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 >> kodikos_kath >> eks >> protimisi >> arithmos_foititwn;
			Lista[i].set_onoma_math(onoma_math);
			Lista[i].set_onoma_kath(onoma_kath);
			Lista[i].set_kodiko_kath(kodikos_kath);
			Lista[i].set_eks(eks);
			Lista[i].set_protimisi(protimisi);
			Lista[i].set_arithmos_foititwn(arithmos_foititwn);
			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() << "kai protimaei na eksetastei tin imera: " <<
Lista[i].get_protimisi() <<" kai tha eksetastoun " << Lista[i].get_arithmos_foititwn() << " foitites." << 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()%5);
			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 xwr_aithousas[5];
		xwr_aithousas[0] = 40;
		xwr_aithousas[1] = 60;
		xwr_aithousas[2] = 100;
		xwr_aithousas[3] = 150;
		xwr_aithousas[4] = 250;
		int temp;
```

```
int oliko_kostos=0;
const int max_sugrouseis=287;
int kostos[num];
for (int i=0; i<num; i++) {
        kostos[i]=0;
}
float arx_pos_sugr;
int max_kost;
float arx_pos_kost;
float fx;

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_kodiko_kath() == Lista[j].get_kodiko_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++;
                                }
                        }
                }
}
for (int i=0;i<num; i++) {
        if (Lista[i].get_arithmos_foititwn() > xwr_aithousas[Arhikos_pinakas[i].get_aithousa()]) {
                Arhikos_pinakas[i].set_sugrousi(1);
                count_sugkrousewn++;
        }
}
for (int i=0; i<num; i++) {
        if (Arhikos_pinakas[i].get_mera() == Lista[i].get_protimisi() ) {
                kostos[i]=0;
```

```
        }
        else if (Arhikos_pinakas[i].get_mera() > Lista[i].get_protimisi() ) {
                kostos[i]=Arhikos_pinakas[i].get_mera() - Lista[i].get_protimisi() ;
        }
        else if (Lista[i].get_protimisi() > Arhikos_pinakas[i].get_mera() ) {
               kostos[i]=(Lista[i].get_protimisi() - Arhikos_pinakas[i].get_mera() );
        }
      }
     for (int i=0; i<num; i++) {
          oliko_kostos = oliko_kostos + kostos[i];
     }
     arx_pos_sugr = count_sugkrousewn*100/max_sugrouseis;
     max_kost=0;
    for (int i=0;i<num; i++){
       if (14 - Lista[i].get_protimisi() >= Lista[i].get_protimisi() ) {
            max_kost=max_kost + 14 - Lista[i].get_protimisi();
       }
       else  { max_kost=max_kost + Lista[i].get_protimisi();
       }
    }
    arx_pos_kost = oliko_kostos*100/max_kost;
    fx = 0.85*arx_pos_sugr + 0.15*arx_pos_kost;
    int max_epanalipseis=0;
    int epoxes1=0;
    int min_sugrouseis=count_sugkrousewn;
    int min_kostos=oliko_kostos;
    int sum_kostos=0;
    int sugrouseis1;
    int temp_mera;
    int temp_ora;
    int temp_aithousa;
    int temp_sugrouseis[num];
    float min_fx=fx;
   float Fx;
   float pos_sugr;
   float pos_kost;
   while ((max_epanalipseis<50) && (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<5;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 s=0;s<num; s++) {
```

```
                    if (Lista[s].get_arithmos_foititwn() > xwr_aithousas[Arhikos_pinakas[s].get_aithousa()]) {
                            Arhikos_pinakas[s].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_kodiko_kath() == Lista[n].get_kodiko_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);
                                    }
                                }
                            }
                        }
                    }
                for (int p=0; p<num; p++) {
                    if (Arhikos_pinakas[p].get_mera() == Lista[p].get_protimisi() ) {
                        kostos[p]=0;
                    }
                    else if (Arhikos_pinakas[p].get_mera() > Lista[p].get_protimisi() ) {
                        kostos[p]=Arhikos_pinakas[p].get_mera() - Lista[p].get_protimisi() ;
                    }
                    else if (Lista[p].get_protimisi() > Arhikos_pinakas[p].get_mera() ) {
                        kostos[p]=(Lista[p].get_protimisi() - Arhikos_pinakas[p].get_mera() );
                    }
                }
                sum_kostos = 0;
                for (int p=0; p<num; p++) {
                    sum_kostos = sum_kostos + kostos[p];
                }
            pos_sugr = sugrouseis1*100/max_sugrouseis;
            pos_kost = sum_kostos*100/max_kost;
            Fx = 0.85*pos_sugr + 0.15*pos_kost;
                if ( sugrouseis1 <= min_sugrouseis) {
                    temp_mera=i;
                    temp_ora=j;
                    temp_aithousa=k;
                    min_sugrouseis=sugrouseis1;
                }
                if (Fx <= min_fx ) {
                    temp_mera=i;
                    temp_ora=j;
                    temp_aithousa=k;
                    min_fx=Fx;
                }
                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 ++;
    }
    cout << "max_epanalipseis: " << max_epanalipseis << endl;
    cout << "sugrouseis: " << min_sugrouseis << endl;
    cout << "pososto sugrousewn: "<< pos_sugr << "%" <<endl;
    cout << "pososto kostous: " << pos_kost<< "%" << endl;
    cout << "Fx: " << Fx << endl;
    return min_sugrouseis;
}

void Eksetastiki::print_programma() {

    ofstream fp;
    fp.open("programma eksetastikis.doc");
                        fp << "DATE                                    |          AITHOUSA |          MATHIMA"<< endl;
                        fp << "_____________________________________________________________________________ "<< endl;



    for(int z=0;z<num; z++){
      if (Arhikos_pinakas[z].get_ora()==0) {
        if (Arhikos_pinakas[z].get_aithousa()==0) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (9  - 12)       |          " << "A                |          " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(9  - 12)       |          " << "A                |          " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==1) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (9  - 12)       |          " << "B                |          " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(9  - 12)       |          " << "B                |          " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==2) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (9  - 12)       |          " << "C                |          " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(9  - 12)       |          " << "C                |          " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==3) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (9  - 12)       |          " << "D                |          " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(9  - 12)       |          " << "D                |          " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==4) {
            if(Arhikos_pinakas[z].get_mera()<10)
```

```
                        fp << Arhikos_pinakas[z].get_mera() << ", (9  - 12)       |          " << "E                |          " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(9  - 12)        |          " << "E                |          " <<
Lista[z].get_onoma_math() << endl;

        }
      }
      else if (Arhikos_pinakas[z].get_ora()==1) {
        if (Arhikos_pinakas[z].get_aithousa()==0) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (12 - 15)       |          " << "A                |          " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(12 - 15)        |          " << "A                |          " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==1) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (12 - 15)       |          " << "B                |          " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(12 - 15)        |          " << "B                |          " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==2) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (12 - 15)       |          " << "C                |          " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(12 - 15)        |          " << "C                |          " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==3) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (12 - 15)       |          " << "D                |          " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(12 - 15)        |          " << "D                |          " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==4) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (12 - 15)       |          " << "E                |          " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(12 - 15)        |          " << "E                |          " <<
Lista[z].get_onoma_math() << endl;

        }
      }
      else if (Arhikos_pinakas[z].get_ora()==2) {
        if (Arhikos_pinakas[z].get_aithousa()==0) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (15 - 18)       |          " << "A                |          " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(15 - 18)        |          " << "A                |          " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==1) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (15 - 18)       |          " << "B                |          " <<
Lista[z].get_onoma_math() << endl;
            else
```

```
                        fp << Arhikos_pinakas[z].get_mera() << ",(15 - 18)      |       " << "B             |       " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==2) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (15 - 18)      |       " << "C             |       " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(15 - 18)      |       " << "C             |       " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==3) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (15 - 18)      |       " << "D             |       " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(15 - 18)      |       " << "D             |       " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==4) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (15 - 18)      |       " << "E             |       " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(15 - 18)      |       " << "E             |       " <<
Lista[z].get_onoma_math() << endl;

        }
      }
      else if (Arhikos_pinakas[z].get_ora()==3) {
        if (Arhikos_pinakas[z].get_aithousa()==0) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (18 - 21)      |       " << "A             |       " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(18 - 21)      |       " << "A             |       " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==1) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (18 - 21)      |       " << "B             |       " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(18 - 21)      |       " << "B             |       " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==2) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (18 - 21)      |       " << "C             |       " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(18 - 21)      |       " << "C             |       " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==3) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (18 - 21)      |       " << "D             |       " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(18 - 21)      |       " << "D             |       " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==4) {
            if(Arhikos_pinakas[z].get_mera()<10)
```

```
                        fp << Arhikos_pinakas[z].get_mera() << ", (18 - 21)        |              " << "E                    |              " <<
Lista[z].get_onoma_math() << endl;
                else
                        fp << Arhikos_pinakas[z].get_mera() << ",(18 - 21)        |              " << "E                    |              " <<
Lista[z].get_onoma_math() << endl;

        }
      }
                }
    fp.close();
}

void Eksetastiki::print_teliko_pinaka() {

for(int z=0; z<num; z++){
   if (Arhikos_pinakas[z].get_ora()==0) {
        if (Arhikos_pinakas[z].get_aithousa()==0) {
          cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 9 ews 12"  << ",stin aithousa A." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==1) {
          cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 9 ews 12"  << ",stin aithousa B." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==2) {
          cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 9 ews 12"  << ",stin aithousa C." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==3) {
          cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 9 ews 12"  << ",stin aithousa D." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==4) {
          cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 9 ews 12"  << ",stin aithousa E." << endl;
        }
    }
    else if (Arhikos_pinakas[z].get_ora()==1) {
        if (Arhikos_pinakas[z].get_aithousa()==0) {
          cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 12 ews 3"  << ",stin aithousa A." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==1) {
          cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 12 ews 3"  << ",stin aithousa B." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==2) {
          cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 12 ews 3"  << ",stin aithousa C." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==3) {
          cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 12 ews 3"  << ",stin aithousa D." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==4) {
          cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 12 ews 3"  << ",stin aithousa E." << endl;
        }
    }
   else if (Arhikos_pinakas[z].get_ora()==2) {
        if (Arhikos_pinakas[z].get_aithousa()==0) {
          cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 3 ews 6"  << ",stin aithousa A." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==1) {
          cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 3 ews 6"  << ",stin aithousa B." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==2) {
          cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 3 ews 6"  << ",stin aithousa C." << endl;
```

```
        }
        else if (Arhikos_pinakas[z].get_aithousa()==3) {
           cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 3 ews 6"  << ",stin aithousa D." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==4) {
          cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 3 ews 6"  << ",stin aithousa E." << endl;
        }
     }
     else if (Arhikos_pinakas[z].get_ora()==3) {
        if (Arhikos_pinakas[z].get_aithousa()==0) {
           cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 6 ews 9"  << ",stin aithousa A." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==1) {
           cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 6 ews 9"  << ",stin aithousa B." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==2) {
           cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 6 ews 9"  << ",stin aithousa C." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==3) {
           cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 6 ews 9"  << ",stin aithousa D." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==4) {
          cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 6 ews 9"  << ",stin aithousa E." << 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 < 50 );
               cout << "epanekkiniseis: " << max_epanekkiniseis-1 << endl;
              if (min_sygrouseis ==0){
                    Ex.print_teliko_pinaka();
                    Ex.print_programma();
               }
               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;
```

ΠΑΡΑΡΤΗΜΑ Γ

PARETO OPTIMALITY

```
#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 kodikos_kath;
		int eks;
		int protimisi;
		int arithmos_foititwn;
public:
		void set_onoma_math(std::string string1){onoma_math = string1;}
		std::string get_onoma_math(){return onoma_math;}
		void set_onoma_kath(std::string string){onoma_kath = string;}
		std::string get_onoma_kath(){return onoma_kath;}
		void set_kodiko_kath(int k){kodikos_kath = k;}
		int get_kodiko_kath(){return kodikos_kath;}
		void set_eks(int e){eks = e;}
		int get_eks(){return eks;}
		void set_protimisi (int pr) {protimisi=pr; }
		int get_protimisi() {return protimisi; }
		void set_arithmos_foititwn(int ar_foit){arithmos_foititwn = ar_foit;}
		int get_arithmos_foititwn(){return arithmos_foititwn;}
};

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 print_programma();
```

```
};

void Eksetastiki::create_lista(){
		std::string onoma_math;
		std::string onoma_kath;
		int kodikos_kath;
		int eks;
		int protimisi;
		int arithmos_foititwn;
		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 >> kodikos_kath >> eks >> protimisi >> arithmos_foititwn;
			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 >> kodikos_kath >> eks >> protimisi >> arithmos_foititwn;
			Lista[i].set_onoma_math(onoma_math);
			Lista[i].set_onoma_kath(onoma_kath);
			Lista[i].set_kodiko_kath(kodikos_kath);
			Lista[i].set_eks(eks);
			Lista[i].set_protimisi(protimisi);
			Lista[i].set_arithmos_foititwn(arithmos_foititwn);
			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() << " protimaei na eksetastei tin imera: " <<
Lista[i].get_protimisi() << " kai tha eksetastoun " << Lista[i].get_arithmos_foititwn() << " foitites." << endl;
		}

 }

void Eksetastiki::create_arhiko_pinaka() {
		int oliko_kostos=0;
		int kostos[num];
		for (int i=0; i<num; i++) {
			kostos[i]=0;
		}
		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);
			Arhikos_pinakas[i].set_aithousa(temp);
			Arhikos_pinakas[i].set_sugrousi(0);
		}
    for (int i=0; i<num; i++) {
       if (Arhikos_pinakas[i].get_mera() == Lista[i].get_protimisi() ) {
            kostos[i]=0;
       }
       else if (Arhikos_pinakas[i].get_mera() > Lista[i].get_protimisi() ) {
```

```
            kostos[i]=Arhikos_pinakas[i].get_mera() - Lista[i].get_protimisi() ;
        }
        else if (Lista[i].get_protimisi() > Arhikos_pinakas[i].get_mera() ) {
            kostos[i]=(Lista[i].get_protimisi() - Arhikos_pinakas[i].get_mera() );
        }
    }
    for (int i=0; i<num; i++) {
     oliko_kostos = oliko_kostos + kostos[i];
    }
}

int Eksetastiki::eleghos_sugrouseon() {
            int count_sugkrousewn=0;
            int count_mathimaton[15];
            int xwr_aithousas[5];
            xwr_aithousas[0] = 40;
            xwr_aithousas[1] = 60;
            xwr_aithousas[2] = 100;
            xwr_aithousas[3] = 150;
            xwr_aithousas[4] = 250;
            int temp;
            int oliko_kostos;
            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;
            }
          int kostos[num];
          for (int i=0; i<num; i++) {
              kostos[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_kodiko_kath() == Lista[j].get_kodiko_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++;
                                        }
                                }
                        }
                }
        }
        for (int i=0;i<num; i++) {
            if (Lista[i].get_arithmos_foititwn() > xwr_aithousas[Arhikos_pinakas[i].get_aithousa()]) {
                    Arhikos_pinakas[i].set_sugrousi(1);
                    count_sugkrousewn++;
            }
        }
        int max_epanalipseis=0;
        int epoxes1=0;
        int min_sugrouseis=count_sugkrousewn;
        int min_kostos=oliko_kostos;
        int sum_kostos=0;
        int sugrouseis1;
        int temp_mera;
        int temp_ora;
        int temp_aithousa;
        int temp_sugrouseis[num];
        int temp_mera1;
        int temp_ora1;
        int temp_aithousa1;
        int c0=0;
        int c1=0;
        while ((max_epanalipseis<50) && (min_sugrouseis!=0)) {
          int pareto_optimal=1;
          for (int s=0;s<num; s++) {
              if (Arhikos_pinakas[s].get_sugrousi()==1) {
                  temp_mera1=Arhikos_pinakas[s].get_mera();
                  temp_ora1=Arhikos_pinakas[s].get_ora();
                  temp_aithousa1=Arhikos_pinakas[s].get_aithousa();
                  for (int i=0;i<15;i++) {
                          for (int j=0;j<4;j++) {
                                  for (int k=0;k<5;k++) {
                                          if (pareto_optimal !=0) {
                                                  Arhikos_pinakas[s].set_mera(i);
                                                  Arhikos_pinakas[s].set_ora(j);
                                                  Arhikos_pinakas[s].set_aithousa(k);
                                                  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){
                                                                          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++;
                                                  }
                                                  if (Arhikos_pinakas[l].get_ora() == Arhikos_pinakas[n].get_ora()) {
                                                      if (Arhikos_pinakas[l].get_aithousa() == Arhikos_pinakas[n].get_aithousa()) {
                                                          sugrouseis1++;
                                                      }
                                                      else if (Lista[l].get_kodiko_kath() == Lista[n].get_kodiko_kath()) {
                                                          sugrouseis1++;
                                                      }
```

```
                                        }
                                    }
                                    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++
                                                }
                                            }
                                        }
                                }
                        }
                        for (int p=0; p<num; p++) {
                                if (Arhikos_pinakas[p].get_mera() == Lista[p].get_protimisi() ) {
                                        kostos[p]=0;
                                }
                                else if (Arhikos_pinakas[p].get_mera() > Lista[p].get_protimisi() ) {
                                        kostos[p]=Arhikos_pinakas[p].get_mera() - Lista[p].get_protimisi() ;
                                }
                                else if (Lista[p].get_protimisi() > Arhikos_pinakas[p].get_mera() ) {
                                        kostos[p]=(Lista[p].get_protimisi() - Arhikos_pinakas[p].get_mera() );
                                }
                        }
                        int sum_kostos=0;
                        for (int p=0; p<num; p++) {
                                sum_kostos = sum_kostos + kostos[p];
                         }
                         for (int s=0;s<num; s++) {
                              if (Lista[s].get_arithmos_foititwn() > xwr_aithousas[Arhikos_pinakas[s].get_aithousa()]) {
                                   sugrouseis1++;
                               }
                         }
                         if ((sugrouseis1>=min_sugrouseis) && (sum_kostos>=min_kostos)) {
                                 pareto_optimal=0;
                        }

                }
               }
              }
             }
             Arhikos_pinakas[s].set_mera(temp_mera1);
             Arhikos_pinakas[s].set_ora(temp_ora1);
             Arhikos_pinakas[s].set_aithousa(temp_aithousa1);
           }
        }
        if (pareto_optimal ==1) {
                cout << "einai" << endl;
                c1 ++;
                ofstream f;
                f.open("pareto_optimal ",ios::app);
                f << "****** vrethike i parakatw pareto optimal katastasi: " << endl;
                for (int pr=0; pr<num; pr++) {
                   f << "To mathima " << Lista[pr].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[pr].get_mera() << " tin
ora " << Arhikos_pinakas[pr].get_ora() << " stin aithousa " << Arhikos_pinakas[pr].get_aithousa() << endl;
                 }
                 f.close();
            }
            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<5;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 s=0;s<num; s++) {
                                    if (Lista[s].get_arithmos_foititwn() > xwr_aithousas[Arhikos_pinakas[s].get_aithousa()]) {
                                            Arhikos_pinakas[s].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_kodiko_kath() == Lista[n].get_kodiko_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);
                                                          }
                                                    }
                                             }
                                         }
                                      }
                                      for (int p=0; p<num; p++) {
                                            if (Arhikos_pinakas[p].get_mera() == Lista[p].get_protimisi() ) {
                                                kostos[p]=0;
                                            }
                                            else if (Arhikos_pinakas[p].get_mera() > Lista[p].get_protimisi() ) {
                                                 kostos[p]=Arhikos_pinakas[p].get_mera() - Lista[p].get_protimisi() ;
                                            }
                                            else if (Lista[p].get_protimisi() > Arhikos_pinakas[p].get_mera() ) {
                                                 kostos[p]=(Lista[p].get_protimisi() - Arhikos_pinakas[p].get_mera() );
                                            }
                                      }
                                     sum_kostos = 0;
                                     for (int p=0; p<num; p++) {
                                            sum_kostos = sum_kostos + kostos[p];
```

```
                                        }
                                        if (sugrouseis1 < min_sugrouseis) {
                                                temp_mera=i;
                                                temp_ora=j;
                                                temp_aithousa=k;
                                                min_sugrouseis=sugrouseis1;
                                        }
                                        if (sugrouseis1 == min_sugrouseis) {
                                             if (sum_kostos < min_kostos) {
                                                  temp_mera=i;
                                                  temp_ora=j;
                                                  temp_aithousa=k;
                                                  min_sugrouseis=sugrouseis1;
                                                 min_kostos=sum_kostos;
                                             }
                                        }
                                        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++;
               }
                cout << "epanalipseis: " << max_epanalipseis << endl;
                cout << "sugrouseis: " << min_sugrouseis << endl;
                cout << "To sinoliko kostos apo tis protimiseis twn kathigitwn einai : " << min_kostos << endl;
                cout << "c0: " << c0 << endl;
                cout << "c1: " << c1 << endl;
               return min_sugrouseis;
        }

void Eksetastiki::print_programma() {

    ofstream fp;
    fp.open("programma eksetastikis.doc");
                        fp << "DATE                                    |       AITHOUSA |       MATHIMA"<< endl;
                        fp << "_____________________________________________________________________________ "<< endl;



    for(int z=0;z<num; z++){
      if (Arhikos_pinakas[z].get_ora()==0) {
        if (Arhikos_pinakas[z].get_aithousa()==0) {
                if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (9  - 12)        |       " << "A                 |       " <<
Lista[z].get_onoma_math() << endl;
                else
                        fp << Arhikos_pinakas[z].get_mera() << ",(9  - 12)        |       " << "A                 |       " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==1) {
                if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (9  - 12)        |       " << "B                 |       " <<
Lista[z].get_onoma_math() << endl;
                else
                        fp << Arhikos_pinakas[z].get_mera() << ",(9  - 12)        |       " << "B                 |       " <<
Lista[z].get_onoma_math() << endl;
```

```
	}
	 else if (Arhikos_pinakas[z].get_aithousa()==2) {
		if(Arhikos_pinakas[z].get_mera()<10)
			fp << Arhikos_pinakas[z].get_mera() << ", (9  - 12)	|	" << "C		|	" <<
Lista[z].get_onoma_math() << endl;
		else
			fp << Arhikos_pinakas[z].get_mera() << ",(9  - 12)	|	" << "C		|	" <<
Lista[z].get_onoma_math() << endl;

	}
	 else if (Arhikos_pinakas[z].get_aithousa()==3) {
		if(Arhikos_pinakas[z].get_mera()<10)
			fp << Arhikos_pinakas[z].get_mera() << ", (9  - 12)	|	" << "D		|	" <<
Lista[z].get_onoma_math() << endl;
		else
			fp << Arhikos_pinakas[z].get_mera() << ",(9  - 12)	|	" << "D		|	" <<
Lista[z].get_onoma_math() << endl;

	}
	 else if (Arhikos_pinakas[z].get_aithousa()==4) {
		if(Arhikos_pinakas[z].get_mera()<10)
			fp << Arhikos_pinakas[z].get_mera() << ", (9  - 12)	|	" << "E		|	" <<
Lista[z].get_onoma_math() << endl;
		else
			fp << Arhikos_pinakas[z].get_mera() << ",(9  - 12)	|	" << "E		|	" <<
Lista[z].get_onoma_math() << endl;

	}
   }
   else if (Arhikos_pinakas[z].get_ora()==1) {
     if (Arhikos_pinakas[z].get_aithousa()==0) {
		if(Arhikos_pinakas[z].get_mera()<10)
			fp << Arhikos_pinakas[z].get_mera() << ", (12 - 15)	|	" << "A		|	" <<
Lista[z].get_onoma_math() << endl;
		else
			fp << Arhikos_pinakas[z].get_mera() << ",(12 - 15)	|	" << "A		|	" <<
Lista[z].get_onoma_math() << endl;

	}
	 else if (Arhikos_pinakas[z].get_aithousa()==1) {
		if(Arhikos_pinakas[z].get_mera()<10)
			fp << Arhikos_pinakas[z].get_mera() << ", (12 - 15)	|	" << "B		|	" <<
Lista[z].get_onoma_math() << endl;
		else
			fp << Arhikos_pinakas[z].get_mera() << ",(12 - 15)	|	" << "B		|	" <<
Lista[z].get_onoma_math() << endl;

	}
	 else if (Arhikos_pinakas[z].get_aithousa()==2) {
		if(Arhikos_pinakas[z].get_mera()<10)
			fp << Arhikos_pinakas[z].get_mera() << ", (12 - 15)	|	" << "C		|	" <<
Lista[z].get_onoma_math() << endl;
		else
			fp << Arhikos_pinakas[z].get_mera() << ",(12 - 15)	|	" << "C		|	" <<
Lista[z].get_onoma_math() << endl;

	}
	 else if (Arhikos_pinakas[z].get_aithousa()==3) {
		if(Arhikos_pinakas[z].get_mera()<10)
			fp << Arhikos_pinakas[z].get_mera() << ", (12 - 15)	|	" << "D		|	" <<
Lista[z].get_onoma_math() << endl;
		else
			fp << Arhikos_pinakas[z].get_mera() << ",(12 - 15)	|	" << "D		|	" <<
Lista[z].get_onoma_math() << endl;

	}
	 else if (Arhikos_pinakas[z].get_aithousa()==4) {
		if(Arhikos_pinakas[z].get_mera()<10)
			fp << Arhikos_pinakas[z].get_mera() << ", (12 - 15)	|	" << "E		|	" <<
Lista[z].get_onoma_math() << endl;
		else
```

```
                        fp << Arhikos_pinakas[z].get_mera() << ",(12 - 15)      |       " << "E             |       " <<
Lista[z].get_onoma_math() << endl;

        }
      }
      else if (Arhikos_pinakas[z].get_ora()==2) {
        if (Arhikos_pinakas[z].get_aithousa()==0) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (15 - 18)      |       " << "A             |       " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(15 - 18)      |       " << "A             |       " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==1) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (15 - 18)      |       " << "B             |       " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(15 - 18)      |       " << "B             |       " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==2) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (15 - 18)      |       " << "C             |       " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(15 - 18)      |       " << "C             |       " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==3) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (15 - 18)      |       " << "D             |       " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(15 - 18)      |       " << "D             |       " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==4) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (15 - 18)      |       " << "E             |       " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(15 - 18)      |       " << "E             |       " <<
Lista[z].get_onoma_math() << endl;

        }
      }
      else if (Arhikos_pinakas[z].get_ora()==3) {
        if (Arhikos_pinakas[z].get_aithousa()==0) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (18 - 21)      |       " << "A             |       " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(18 - 21)      |       " << "A             |       " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==1) {
            if(Arhikos_pinakas[z].get_mera()<10)
                        fp << Arhikos_pinakas[z].get_mera() << ", (18 - 21)      |       " << "B             |       " <<
Lista[z].get_onoma_math() << endl;
            else
                        fp << Arhikos_pinakas[z].get_mera() << ",(18 - 21)      |       " << "B             |       " <<
Lista[z].get_onoma_math() << endl;

        }
```

```
        else if (Arhikos_pinakas[z].get_aithousa()==2) {
            if(Arhikos_pinakas[z].get_mera()<10)
                fp << Arhikos_pinakas[z].get_mera() << ", (18 - 21)       |          " << "C                |          " <<
Lista[z].get_onoma_math() << endl;
            else
                fp << Arhikos_pinakas[z].get_mera() << ",(18 - 21)       |          " << "C                |          " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==3) {
            if(Arhikos_pinakas[z].get_mera()<10)
                fp << Arhikos_pinakas[z].get_mera() << ", (18 - 21)       |          " << "D                |          " <<
Lista[z].get_onoma_math() << endl;
            else
                fp << Arhikos_pinakas[z].get_mera() << ",(18 - 21)       |          " << "D                |          " <<
Lista[z].get_onoma_math() << endl;

        }
        else if (Arhikos_pinakas[z].get_aithousa()==4) {
            if(Arhikos_pinakas[z].get_mera()<10)
                fp << Arhikos_pinakas[z].get_mera() << ", (18 - 21)       |          " << "E                |          " <<
Lista[z].get_onoma_math() << endl;
            else
                fp << Arhikos_pinakas[z].get_mera() << ",(18 - 21)       |          " << "E                |          " <<
Lista[z].get_onoma_math() << endl;

        }
      }
              }
    fp.close();
}
void Eksetastiki::print_teliko_pinaka() {

for(int z=0; z<num; z++){
   if (Arhikos_pinakas[z].get_ora()==0) {
        if (Arhikos_pinakas[z].get_aithousa()==0) {
          cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 9 ews 12"  << ",stin aithousa A." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==1) {
           cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 9 ews 12"  << ",stin aithousa B." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==2) {
           cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 9 ews 12"  << ",stin aithousa C." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==3) {
           cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 9 ews 12"  << ",stin aithousa D." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==4) {
          cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 9 ews 12"  << ",stin aithousa E." << endl;
        }
    }
    else if (Arhikos_pinakas[z].get_ora()==1) {
        if (Arhikos_pinakas[z].get_aithousa()==0) {
           cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 12 ews 3"  << ",stin aithousa A." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==1) {
           cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 12 ews 3"  << ",stin aithousa B." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==2) {
           cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 12 ews 3"  << ",stin aithousa C." << endl;
        }
        else if (Arhikos_pinakas[z].get_aithousa()==3) {
```

```
            cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 12 ews 3"  << ",stin aithousa D." << endl;
          }
          else if (Arhikos_pinakas[z].get_aithousa()==4) {
            cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 12 ews 3"  << ",stin aithousa E." << endl;
          }
      }
     else if (Arhikos_pinakas[z].get_ora()==2) {
          if (Arhikos_pinakas[z].get_aithousa()==0) {
            cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 3 ews 6"  << ",stin aithousa A." << endl;
          }
          else if (Arhikos_pinakas[z].get_aithousa()==1) {
             cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 3 ews 6"  << ",stin aithousa B." << endl;
          }
          else if (Arhikos_pinakas[z].get_aithousa()==2) {
             cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 3 ews 6"  << ",stin aithousa C." << endl;
          }
          else if (Arhikos_pinakas[z].get_aithousa()==3) {
             cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 3 ews 6"  << ",stin aithousa D." << endl;
          }
          else if (Arhikos_pinakas[z].get_aithousa()==4) {
            cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 3 ews 6"  << ",stin aithousa E." << endl;
          }
      }
     else if (Arhikos_pinakas[z].get_ora()==3) {
          if (Arhikos_pinakas[z].get_aithousa()==0) {
             cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 6 ews 9"  << ",stin aithousa A." << endl;
          }
          else if (Arhikos_pinakas[z].get_aithousa()==1) {
             cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 6 ews 9"  << ",stin aithousa B." << endl;
          }
          else if (Arhikos_pinakas[z].get_aithousa()==2) {
             cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 6 ews 9"  << ",stin aithousa C." << endl;
          }
          else if (Arhikos_pinakas[z].get_aithousa()==3) {
             cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 6 ews 9"  << ",stin aithousa D." << endl;
          }
          else if (Arhikos_pinakas[z].get_aithousa()==4) {
            cout << "To mathima " << Lista[z].get_onoma_math() << " tha ginei tin mera " << Arhikos_pinakas[z].get_mera() <<
",ora 6 ews 9"  << ",stin aithousa E." << 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 < 50 );
            cout << "epanekkiniseis: " << max_epanekkiniseis-1 << endl;
            if (min_sygrouseis ==0){
                  Ex.print_programma();
                 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;
}
```