ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

Μεταπτυχιακή Εργασία

ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ ΑΝΑΘΕΣΗΣ ΙΠΤΑΜΕΝΩΝ ΣΕ ΕΚΠΑΙΔΕΥΤΙΚΕΣ ΣΥΝΕΔΡΙΕΣ ΜΕ ΧΡΗΣΗ ΜΕΙΚΤΟΥ ΑΚΕΡΑΙΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

υπό

ΜΑΥΡΑΝΤΖΑ ΔΕΣΠΟΙΝΑ

Διπλωματούχου Μηχανολόγου Μηχανικού Α.Π.Θ., 2012

Επιβλέπων Καθηγητής: Κοζανίδης Γεώργιος

Υπεβλήθη για την εκπλήρωση μέρους των

απαιτήσεων για την απόκτηση του

Μεταπτυχιακού Διπλώματος Ειδίκευσης

2014

Εγκρίθηκε από τα Μέλη της Τριμελούς Εξεταστικής Επιτροπής:

Πρώτος Εξεταστής (Επιβλέπων)	Δρ. Γεώργιος Κοζανίδης Επίκουρος Καθηγητής, Τμήμα Μηχανολόγων Μηχανικών, Πανεπιστήμιο Θεσσαλίας
Δεύτερος Εξεταστής	Δρ. Γεώργιος Λυμπερόπουλος Καθηγητής, Τμήμα Μηχανολόγων Μηχανικών, Πανεπιστήμιο Θεσσαλίας
Τρίτος Εξεταστής	Δρ. Δημήτριος Παντελής Επίκουρος Καθηγητής, Τμήμα Μηχανολόγων Μηχανικών, Πανεπιστήμιο Θεσσαλίας

ΕΥΧΑΡΙΣΤΙΕΣ

Στο σημείο αυτό θα ήθελα να ευχαριστήσω θερμά όλους τους ανθρώπους που με άμεσο ή έμμεσο τρόπο συνέβαλαν στην ολοκλήρωση της μεταπτυχιακής εργασίας.

Πιο συγκεκριμένα ευχαριστώ θερμά τον επιβλέποντα καθηγητή κ. Γεώργιο Κοζανίδη, Επίκουρο Καθηγητή του τμήματος Μηχανολόγων Μηχανικών της Πολυτεχνικής σχολής του Πανεπιστημίου Θεσσαλίας, ο οποίος μου έδωσε την ευκαιρία να ασχοληθώ με ένα υπαρκτό πρόβλημα από την περιοχή των αερομεταφορών. Επίσης, χωρίς την καθοδήγηση του σε θέματα προγραμματισμού και χωρίς τις πολύτιμες συμβουλές και γνώσεις η εργασία αυτή δεν θα είχε υλοποιηθεί.

Επίσης, ευχαριστώ και τα υπόλοιπα δύο μέλη της εξεταστικής επιτροπής της μεταπτυχιακής εργασίας μου, κ. Γεώργιο Λυμπερόπουλο Καθηγητή του Τμήματος Μηχανολόγων Μηχανικών και Δημήτριο Παντελή Επίκουρο Καθηγητή του ίδιου τμήματος για τον χρόνο που αφιέρωσαν για την ανάγνωση της εργασίας

Τέλος, θα ήθελα να ευχαριστήσω πάνω από όλους τα μέλη της οικογένεια μου για τη συμπαράσταση, την αγάπη και την κατανόηση που έχουν δείξει σε όλη τη διάρκεια των σπουδών μου.

ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ ΑΝΑΘΕΣΗΣ ΙΠΤΑΜΕΝΩΝ ΣΕ ΕΚΠΑΙΔΕΥΤΙΚΕΣ ΣΥΝΕΔΡΙΕΣ ΜΕ ΧΡΗΣΗ ΜΕΙΚΤΟΥ ΑΚΕΡΑΙΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

ΜΑΥΡΑΝΤΖΑ ΔΕΣΠΟΙΝΑ

Πανεπιστήμιο θεσσαλίας, Τμήμα Μηχανολόγων Μηχανικών

Επιβλέπων Καθηγητής Δρ. Γεώργιος Κοζανίδης

Επίκουρος Καθηγητής, Τμήμα Μηχανολόγων Μηχανικών, Πανεπιστήμιο Θεσσαλίας

Περίληψη

Στην σύγχρονη εποχή, οι αεροπορικές εταιρείες δείχνουν ιδιαίτερο ενδιαφέρον στην εκπαίδευση του ιπτάμενου προσωπικού τους, κυρίως επειδή τα προσόντα τους και οι ικανότητές τους διασφαλίζουν την ασφάλεια των αερομεταφορών. Γίνεται λοιπόν φανερό πως ο αποτελεσματικός προγραμματισμός της εκπαίδευσης του ιπτάμενου προσωπικού αποτελεί ένα σημαντικό πρόβλημα των αεροπορικών εταιρειών.

Σε αυτή την εργασία παρουσιάζουμε ένα μοντέλο το οποίο βελτιστοποιεί την ανάθεση των ιπταμένων σε εκαιδευτικές συνεδρίες, το οποίο στηρίζεται στις αρχές του μεικτού ακέραιου προγραμματισμού. Στο μοντέλο αυτό εισάγουμε τα χαρακτηριστικά κάθε ιπταμένου και κάθε συνεδρίας, καθώς και περιορισμούς που προκύπτουν από τη λειτουργία του συστήματος. Τέτοιοι είναι οι περιορισμοί χωρητικότητας των συνεδριών,όπως και οι περιορισμοί γλώσσας καθώς οι ιπτάμενοι που θα ανατεθούν σε μια συνεδρία θα πρέπει να ομιλούν την ίδια γλώσσα. Επίσης, εξίσου σημαντικοί είναι και οι περιορισμοί αρχαιότητας που υποδηλώνουν πως θα πρέπει να προσπαθούμε να ικανοποιήσουμε την πρώτη προτίμηση κάθε ιπταμένου με τήρηση αυστηρής αρχαιότητας. Ο βασικός σκοπός αυτού του μοντέλου είναι η ελαχιστοποίηση των ιπταμένων που δεν θα ανατεθούν σε εκπαιδευτικές συνεδρίες, τηρώντας όμως περιορισμούς αυστηρής αρχαιότητας.

Λέξεις κλειδιά: ιπτάμενο προσωπικό, εκπαιδευτικές συνεδρίες, βελτιστοποίηση, πρόβλημα ανάθεσης, μεικτός ακέραιος προγραμματισμός, αυστηρή αρχαιότητα.

MIXED INTEGER PROGRAMMING FOR OPTIMAL ASSIGNMENT OF FLIGHT CREW PERSONNEL TO TRAINING SESSIONS

MAVRANTZA DESPOINA

Uiversity of Thessaly, Department of Mechanical Engineering

Supervisor: Dr. George Kozanidis, Assistant Professor ,Department of Mechanical Engineering, University of Thessaly

Abstract

Nowadays, the training of the flight crew is a crucial issue for airlines because the skills and the qualifications of these specialists are the main guarantees of the efficiency and the security of aviation. Thus, it's obvious that assigning crew personnel to recurrent training is a very important and complicated problem for airlines.

In this thesis, we represent a mathematical model for optimal assignment of crew personnel to training sessions, which is based on a mixed integer programming formulation.The problem is highly complex, because we take into account the characteristics of each crew member, the sessions' attributes, as well as various operational constraints. These are capacity constraints for each training classroom, as well as language constraints, since the crew members that will be assigned to the same classroom must speak the same language. Last but not least, there are seniority constraints, which dictate that special caution should be given to maximizing the satisfaction of the preferences of the more senior crew members. The objective of the model is to minimize the number of crew members that remain unassigned, which, in turn, is subject to strict seniority restrictions.

Key words: crew personnel, training sessions,optimization, assignment problem, mixed integer programming, strict seniority.

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΕΡΙΕΧΟΜΕΝΑ ΠΙΝΑΚΩΝ

ΠΕΡΙΕΧΟΜΕΝΑ ΕΙΚΟΝΩΝ

ΠΡΟΛΟΓΟΣ

1. Εισαγωγή

1.1 Σκοπός εργασίας

Στις μέρες μας, οι αεροπορικές εταιρείες χρησιμοποιούν μια μεγάλη ποικιλία εργαλείων και λογισμικών για την αποτελεσματικότερη λειτουργία τους. Η όλη διαδικασία προγραμματισμού που περιλαμβάνει την κατασκευή χρονοδιαγραμμάτων, τον προγραμματισμό του προσωπικού, τον προγραμματισμό των πτήσεων, κ.α. υποστηρίζεται από μια μεγάλη γκάμα λογισμικών. Εκτός από το γεγονός ότι αυτά τα λογισμικά πρέπει να είναι αξιόπιστα, πρέπει επίσης να προσφέρουν βέλτιστες λύσεις στα προβλήματα του συστήματος σε σύντομο χρονικό διάστημα.

Λόγω του ανταγωνιστικού περιβάλλοντος στο οποίο λειτουργούν, οι αεροπορικές εταιρείες δείχνουν αμείωτο ενδιαφέρον για τέτοιου είδους λογισμικά και εφαρμογές που μπορούν να βοηθήσουν στη βέλτιστη αξιοποίηση των πόρων τους με ταυτόχρονη μείωση του λειτουργικού τους κόστους. Η πρόβλεψη των πιλότων που απαιτούνται σε διάφορες θέσεις, ο προσδιορισμός του χρόνου που πραγματοποιείται η εκπαίδευση του προσωπικού αλλά και ο προσδιορισμός του προσωπικού που θα συμμετέχει σε αυτήν είναι μερικά από τα σοβαρά προβλήματα που αντιμετωπίζουν οι αεροπορικές εταιρείες.

Με το πέρασμα των χρόνων, τα συστήματα ελέγχου πτήσης γίνονται όλο και πιο αξιόπιστα όποτε τις περισσότερες φορές τα ατυχήματα οφείλονται σε ανθρώπινο λάθος. Σε περίπτωση όπου συμβεί κάποιο αναπάντεχο γεγονός κατά τη διάρκεια της πτήσης οι πιλότοι που συνεργάζονται με τους εναέριους ελεγκτές κίνησής θα πρέπει να ξέρουν ακριβώς τι πρέπει να κάνουν. Επίσης το ιπτάμενο προσωπικό καμπίνας θα πρέπει να είναι προετοιμασμένο για οποιοδήποτε είδους προβλήματος μπορεί να προκύψει κατά τη διάρκεια της πτήσης. Για αυτό το λόγο .με αυτή την εργασία δίνουμε έμφαση στον βέλτιστο προγραμματισμό του προσωπικού σε εκπαίδευση, ο οποίος μπορεί να οδηγήσει σε μείωση του κόστους αλλά και στη χρήση πιο προηγμένων τεχνικών εκπαίδευσης. Η κατανομή και ο προγραμματισμός του ανθρώπινου δυναμικού είναι ένα από τα πιο σημαντικά και απαιτητικά θέματα που έχουν να αντιμετωπίσουν οι σύγχρονες αεροπορικές εταιρείες.

1.2. Βιβλιογραφική ανασκόπηση

Όσον αφορά τον προγραμματισμό του ιπτάμενου προσωπικού για εκπαίδευση αλλά και τον βέλτιστο προγραμματισμό αυτών, έχουν γίνει αρκετές μελέτες και έχουν

κατασκευαστεί μαθηματικά μοντέλα η επίλυση των οποίων οδήγησε σε μείωση του λειτουργικού κόστους των αεροπορικών εταιρειών που τα εφάρμοσαν.

Πιο συγκεκριμένα ο M. Shapiro[1] σε έρευνά του που δημοσιεύθηκε τον Ιούνιο του 1981 περιγράφει ένα υπολογιστικό μοντέλο το οποίο ανέπτυξε ο ίδιος για λογαριασμό της εταιρείας AmericanAirlines. Τα αποτελέσματα της έρευνας έδειξαν πως υπάρχει μείωση του κόστους εκπαίδευσης όταν αυτή πραγματοποιείται μετά την απομάκρυνσή των ιπταμένων από το πρόγραμμα πτήσεων και όχι κατά το ρεπό τους. Το υπολογιστικό αυτό μοντέλο, το οποίο χρησιμοποιείται ήδη εδώ και μερικά χρόνια από την AmericanAirlines, οδήγησε σε μία σημαντική εξοικονόμηση πόρων της εταιρείας της τάξης των $250,000 ετησίως.

Αρκετά χρόνια αργότερα, το 1995, ο R.P. Brown[2]στη διατριβή του αναπτύσσει ένα μοντέλο δι-κριτήριου μεικτού ακέραιου προγραμματισμού για το σώμα πεζοναυτών, με το οποίο ποσοτικοποιεί την ετοιμότητα των πεζοναυτών στη μάχη με βάση τον προγραμματισμό εκπαίδευσής τους. Επίσης, εκτός από την ετοιμότητα των πεζοναυτών, το μοντέλο αυτό λαμβάνει υπόψη του και το γεγονός ότι όλοι οι πεζοναύτες θα πρέπει να έχουν ίσες ευκαιρίες. Επομένως, το μοντέλο μεγιστοποιεί μια συνάρτηση που περιλαμβάνει την ετοιμότητα των πεζοναυτών αλλά και την δίκαιη κατανομή εργασιών για ένα ορίζοντα ενενήντα ημερών.

Οι GangYu, S. Dugan και M. Argüello[4] σε έρευνα τους το 1998 περιγράφουν ένα ολοκληρωμένο σύστημα λήψης αποφάσεων για την ανάθεση πιλότων σε εκπαιδευτικές συνεδρίες.Μάλιστα, η εφαρμογή ενός προτύπου που βασίζεται στην ευρετική λύση αυτού του μοντέλου από την εταιρεία ContinentalAirlinesτης απέφερε εξοικονόμηση πόρων της τάξης των 6 εκατομμυρίων δολαρίων ετησίως μόνο από τον αποτελεσματικότερο προγραμματισμό εκπαίδευσης των πιλότων της.

Το 2003 ο AslanDavut [12]στη διατριβή του αναπτύσσει ένα εργαλείο λήψης αποφάσεων που χρησιμοποιεί έναν ευρετικό αλγόριθμο με σκοπό τον προγραμματισμό των εκπαιδευτικών πτήσεων της μοίρας αξιοποιώντας τους διαθέσιμους πόρους και τηρώντας τα αυστηρά χρονοδιαγράμματα.

Μια ακόμα μελέτη από τους Sohomi, Bailey, Martin, Carterκαι Johnson[6] αφορούσε την ανάθεση των πιλότων της εταιρείας DeltaAirlinesπρος εκπαίδευση. Στην μελέτη αυτή ανέπτυξαν ένα αυτοματοποιημένο σύστημα βελτιστοποίησης το οποίο προγραμματίζει την εκπαίδευση σύμφωνα με τις απαιτήσεις του κάθε πιλότου με πρωταρχικό σκοπό την ελαχιστοποίηση του κόστους εκπαίδευσης και τη μεγιστοποίηση των εκπαιδευτικών αναθέσεων.

Το 2004 οι XiangtongQi, J.F. Bard και GangYu [8] δημοσίευσαν την εργασία τους με θέμα τον προγραμματισμό των πιλότων για εκπαίδευση στις εγκαταστάσεις της ContinentalAirlines. Όταν οι πιλότοι ανατίθενται σε νέα καθήκοντα θα πρέπει πρώτα

να περάσουν από εκπαίδευση οκτώ εβδομάδων. Κατά τη διάρκεια αυτών των εβδομάδων δεν μπορούν να εργαστούν και αυτό δημιουργεί ένα επιπλέον κόστος για την εταιρεία. Σκοπός λοιπόν της έρευνάς τους ήταν η δημιουργία ενός μοντέλου που στοχεύει στη μείωση της διάρκειας όλων των προγραμμάτων εκπαίδευσης. Οι λύσεις προκύπτουν από τη χρήση ενός branch-and-bound αλγορίθμου και μιας οικογένειας ευρετικών αλγορίθμων που βασίζονται στην ιδέα ενός κυλιόμενου χρονικού ορίζοντα. Έπειτα, πραγματοποίησαν μια σειρά υπολογιστικών πειραμάτων προκειμένου να εξετάσουν την αξιοπιστία αυτών των αλγορίθμων οι οποίοι εφαρμόστηκαν με επιτυχία από την ContinentalAirlines.

Η έρευνα των B. Thengvall και XiangtongQi[5] που δημοσιεύτηκε το 2007 χρησιμοποιεί ένα μοντέλο μεικτού ακέραιου προγραμματισμού και έναν αλγόριθμο branch-and-boundγια την κατασκευή χρονοδιαγραμμάτων ελάχιστης διάρκειας για την εκπαίδευση πιλότων.

1.3. Οργάνωση της εργασίας

Η εργασία αποτελείται από τέσσερα μέρη: την εισαγωγή, το θεωρητικό μέρος, το ερευνητικό μέρος και τον επίλογο. Πιο συγκεκριμένα:

- Η εισαγωγή περιλαμβάνει τρεις υποενότητες. Πρώτη είναι εκείνη όπου περιγράφει το λόγο για τον οποίο ασχοληθήκαμε με το συγκεκριμένο θέμα και αναλύειτο σκοπό της εργασίας. Στην επόμενη παρουσιάζεται η υπάρχουσα βιβλιογραφία, και έπειτα στην τελευταία υποενότητα αυτού του μέρους δίνεται η δομή της εργασίας.
- Το θεωρητικό μέρος αποτελείται από ένα κεφάλαιο το οποίο όπως γίνεται φανερό περιγράφει της βασικές έννοιες της επιχειρησιακής έρευνας και του μεικτού ακέραιου προγραμματισμού.
- Το ερευνητικό μέρος αποτελείται από τέσσερα βασικά κεφάλαια. Στο πρώτο από αυτά περιγράφουμε το υπό μελέτη σύστημα και παρουσιάζουμε τις βασικές συνιστώσες του. Στο δεύτερο αναπτύσσουμε το μαθηματικό μοντέλο δημιουργώντας την αντικειμενική συνάρτηση και τις μαθηματικές σχέσεις που αναπαριστούν τους λειτουργικούς περιορισμούς του συστήματος μας. Στο τρίτο παρουσιάζουμε τις βασικές συναρτήσεις του κώδικα που δημιουργήσαμε με τη γλώσσα προγραμματισμού C. Ενώ στο τελευταίο κεφάλαιο αυτού του μέρους επιλύουμε με τη βοήθεια του κώδικα δύο μικρά προβλήματα και παρουσιάζουμε τα αποτελέσματα από μια σειρά πειραμάτων που πραγματοποιήσαμε.

- Ο επίλογος αποτελεί το τελευταίο μέρος αυτής της μελέτης. Εδώ, κάνουμε μια σύντομη ανασκόπηση της εργασίας και δίνουμε τα συμπεράσματα στα οποία καταλήγουμε. Το κομμάτι αυτό ολοκληρώνεται με προτάσεις οι οποίες θα μπορούσαν να αποτελέσουν κίνητρο για μελλοντικές μελέτες.

ΘΕΩΡΗΤΙΚΟ ΜΕΡΟΣ

2.Θεωρητικό υπόβαθρο

2.1.Ορισμός Επιχειρησιακής Έρευνας

Η Επιχειρησιακή Έρευνα είναι επιστημονικός κλάδος που ασχολείται με την ανάπτυξη μαθηματικών μοντέλων για την περιγραφή συστημάτων και διαδικασιών με κύριο σκοπό την βελτστοποίηση τους και τη λήψη αποφάσεων.Πιο συγκεκριμένα, η Επιχειρησιακή Έρευνα αφορά την εφαρμογή μαθηματικών (ποσοτικών) τεχνικών για τη λήψη αποφάσεων. Στην Επιχειρησιακή Έρευνα ένα πρόβλημα αρχικά προσδιορίζεται πλήρως και παρουσιάζεται ως ένα μαθηματικό μοντέλο που αποτελείται από μαθηματικές εξισώσεις. Στη συνέχεια υποβάλλεται σε διεξοδική υπολογιστική ανάλυση προκειμένου να πάρουμε μία λύση, η οποία μετά δοκιμάζεται σε πραγματικές καταστάσεις μέχρις ότου να βρεθεί η βέλτιστη λύση για το πρόβλημα.Ο όρος ‘’επιχειρησιακή’’ έχει την έννοια της διαδικασίας λειτουργίας και όχι της εταιρείας.

2.2.Ιστορική αναδρομή

Η Επιχειρησιακή Έρευνα ασχολείται με την ανάλυση προβλημάτων και τη λήψη αποφάσεων. Η αρχή της Επιχειρησιακής ¨Έρευνας ως επιστήμη μπορεί να προσδιοριστεί στην Αγγλία στα μέσα της δεκαετίας του ’30. Λίγο πριν το ξέσπασμα του Β Παγκοσμίου Πολέμου, η αεροπορική δύναμη της Αγγλίας προετοιμαζόταν για την αντιμετώπιση της εναέριας Γερμανικής εισβολής. Με την πρόσφατά ανεπτυγμένη τεχνολογία του ραντάρ, οι Βρετανοί είχαν την ανάγκη να σχεδιάσουν τη στρατηγική τους για τον εντοπισμό και την αναχαίτιση των γερμανικών πολεμικών αεροσκαφών.
Μετά την ολοκλήρωση του πολέμου, η Επιχειρησιακή Έρευνα ακολούθησε το δρόμο της και απέκτησε ευρύτερο πεδίο εφαρμογής, ανάλογο με τις ανάγκες της ανθρωπότητας σε καιρό ειρήνης. Με τη ραγδαία βιομηχανική ανάπτυξη κατά τη δεκαετία του ’50 ήρθαν στην επιφάνεια πολύπλοκα προβλήματα διοίκησης, παραγωγής και εμπορίου προϊόντων. Τα προβλήματα αυτά ήταν όμοια με αυτά που αντιμετώπιζαν οι πολεμικές βάσεις κατά τη διάρκεια του πολέμου με μόνη διαφορά ότι το πεδίο εφαρμογής τους είχε αλλάξει. Με το πέρασμα των χρόνων η Επιχειρησιακή Έρευνα καθιερώθηκε σε επιχειρήσεις, βιομηχανίες και οργανισμούς. Έπειτα, η εμφάνιση των ηλεκτρονικών υπολογιστών έδωσε ακόμη μεγαλύτερη ώθηση στην ανάπτυξη αυτού του επιστημονικού πεδίου.

2.3.Η προσέγγιση της Επιχειρησιακής Έρευνας στη λήψη βέλτιστων αποφάσεων

Στην Επιχειρησιακή Έρευνα με τον όρο σύστημα εννοούμε πάσης φύσεως επιχειρήσεις, βιομηχανίες, κρατικούς οργανισμούς κλπ. Για την αντιμετώπιση των προβλημάτων ενός συστήματος είναι απαραίτητη η αναπαράστασή του με ένα μαθηματικό μοντέλο. Δηλαδή πρέπει να περιγράψουμε τις σημαντικές σχέσεις μεταξύ των χαρακτηριστικών του συστήματος με παρόμοιες σχέσεις μεταξύ μαθηματικών στοιχείων. Όπως γίνεται φανερό είναι εξαιρετικά δύσκολο το μοντέλο να αναπαριστά με λεπτομέρεια κάθε πτυχή του συστήματος. Ένα ακριβές μοντέλο αναπαριστά ικανοποιητικά όλα τα στοιχεία τα οποία εκτιμάται ότι είναι απαραίτητα για τη λήψη «κάποιας» απόφασης για την επίλυση του προβλήματος, ενώ δεν περιέχει άσχετες με αυτό λεπτομέρειες ή υποθέσεις.

Τα βήματα που ακολουθούνται για την εύρεση της βέλτιστης λύσης είναι:

- Ορισμός του προβλήματος και συλλογή δεδομένων. Στη φάση αυτή αναγνωρίζονται οι βασικές συνιστώσες του προβλήματος, καθορίζονται οι επιθυμητοί στόχοι, και εντοπίζονται οι περιορισμοί που επιβάλλονται από τη λειτουργία του συστήματος.
- Ανάπτυξη του μαθηματικού μοντέλου. Σε αυτό το στάδιο αναπαριστούμε τις βασικές συνιστώσες του προβλήματος με μαθηματικές σχέσεις. Εισάγουμε τις μεταβλητές απόφασης, τις παραμέτρους, τους μαθηματικούς περιορισμούς αλλά και την αντικειμενική συνάρτηση.
- Επίλυση του μαθηματικού μοντέλου. Σε αυτή τη φάση γίνεται προσπάθεια να προσδιοριστούν οι τιμές των μεταβλητών απόφασης έτσι ώστε να τηρούνται οι λειτουργικοί περιορισμοί και να βελτιώνεται η λειτουργία του συστήματος.
- Εφαρμογή της προτεινόμενης λύσης στο υπό μελέτη σύστημα και αξιολόγησή της. Στο τελικό στάδιο παίρνουμε τα αποτελέσματα και τα εφαρμόζουμε στο πραγματικό σύστημα προκειμένου να διαπιστώσουμε αν μπορούν να εφαρμοστούν και αν έχουν νόημα. Πολλές φορές παραλείπονται κάποιες συνιστώσες του προβλήματος οι οποίες μπορεί να φαίνονται ασήμαντες άλλα να είναι εκείνες που να οδηγούν στην απόρριψη της λύσης που προέκυψε από την επίλυση του μαθηματικού μοντέλου.

2.4.Μεικτός ακέραιος γραμμικός προγραμματισμός

Με το πέρασμα των χρόνων έχουν αναπτυχθεί διάφορες τεχνικές για την επίλυση προβλημάτων Επιχειρησιακής Έρευνας. Ένα από τα εργαλεία που έχει αναπτυχθεί και το οποίο χρησιμοποιούμε στην παρούσα εργασία για την επίλυση του

προβλήματος βελτιστοποίησης είναι ο *Μεικτός Ακέραιος Γραμμικός Προγραμματισμός*. Ένα μοντέλο *Γραμμικού Προγραμματισμού* αποτελείται από γραμμικές συναρτήσεις και περιορισμούς, γεγονός που σημαίνει ότι οι μεταβλητές του μοντέλου έχουν μεταξύ τους αναλογικές σχέσεις.

Ένα πρόβλημα στο οποίο μόνο μερικές από τις μεταβλητές απόφασης πρέπει να έχουν ακέραιες τιμές ονομάζεται πρόβλημα *Μεικτού Ακέραιου Προγραμματισμού*.

Άλλες μαθηματικές τεχνικές που χρησιμοποιούνται για την ανάπτυξη μαθηματικών μοντέλων είναι ο Μη Γραμμικός Προγραμματισμός, ο Ακέραιος Προγραμματισμός, ο Δυναμικός Προγραμματισμός, η Θεωρία Παιγνίων κ.α.

ΕΡΕΥΝΗΤΙΚΟ ΜΕΡΟΣ

3. Περιγραφή προβλήματος

Όσο τα αεροπλάνα δεν μπορούν να πετάξουν χωρίς πιλότο, χωρίς πλήρωμα και χωρίς προσωπικό εδάφους, οι ικανότητες, τα προσόντα και η εκπαίδευση αυτών των ανθρώπων εξακολουθούν να αποτελούν την εγγύηση για την αποτελεσματικότητα και την ασφάλεια των αερομεταφορών. Για το λόγο αυτό είναι πολύ σημαντική η συνεχής εκπαίδευση του ιπτάμενου προσωπικού έτσι ώστε να πιστοποιείται η ικανότητα πτήσης τους αλλά και να διασφαλίζεται πως πληρούν όλα τα κριτήρια που απαιτούνται από την Διεθνή Υπηρεσία Αερομεταφορών (IATA = International Air Transport Association). Επίσης, η εκπαίδευσή τους με όλο και πιο σύγχρονες μεθόδους διασφαλίζει πως το ιπτάμενο προσωπικό θα είναι έτοιμο να αντιμετωπίσει οποιαδήποτε κατάσταση κατά τη διάρκεια της πτήσης.
Επομένως, κατανοούμε πως το κόστος εκπαίδευσης του ιπτάμενου προσωπικού αποτελεί ένα σημαντικό κομμάτι του συνολικού λειτουργικού κόστους της κάθε αεροπορικής εταιρείας. Επίσης, κατά τη σχεδίαση της εκπαίδευσης η αεροπορική εταιρεία θα πρέπει να λάβει υπόψη της τη χρονική διάρκεια της εκπαίδευσης, το μέρος όπου θα πραγματοποιηθεί αλλά και το προσωπικό που θα λάβει μέρος σε αυτή χωρίς να υπάρχουν ελλείψεις προσωπικού στην καθημερινή λειτουργία της εταιρείας.

3.1. Συνιστώσες του προβλήματος

Παρατηρούμε λοιπόν ότι το πρόγραμμα εκπαίδευσης του ιπτάμενου προσωπικού είναι αρκετά περίπλοκο και δεν μπορεί να προκύψει από απλούς υπολογισμούς. Για αυτό το λόγο καταγράφουμε όλες τις συνιστώσες του προβλήματος και τις ποσοτικοποιούμε. Τα δεδομένα εισόδου που είναι απαραίτητα για την επίλυση του προβλήματος είναι :

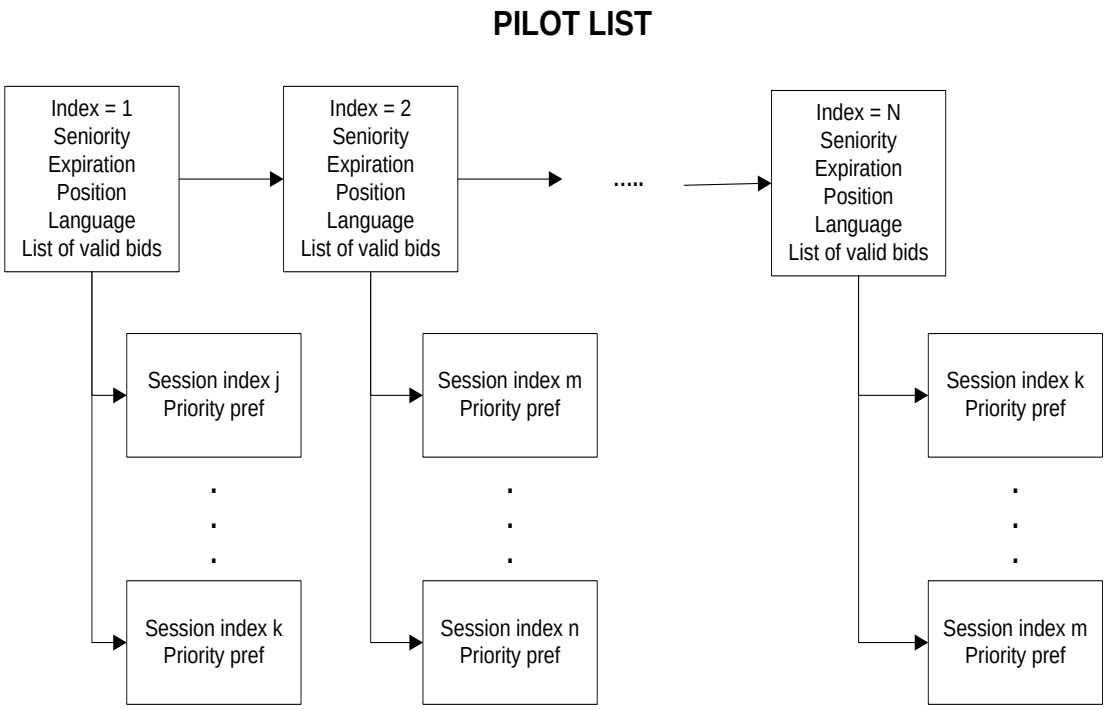
Η λίστα του ιπτάμενου προσωπικού που θα πρέπει να ανατεθεί στις εκπαιδευτικές συνεδρίες. Κάθε μέλος αυτή της λίστας έχει ορισμένα χαρακτηριστικά τα οποία είναι:

- **Θέση (position)**. Σε κάθε εκπαιδευτική συνεδρία υπάρχουν θέσεις που προορίζονται για τους κυβερνήτες, τους συγκυβερνήτες και το προσωπικό καμπίνας. Οι δύο πρώτοι ανήκουν στην κατηγορία του προσωπικού πιλοτηρίου ή αλλιώς των PNT ‘s (PersonnelNavigantTechnique) ενώ οι τελευταίοι ανήκουν στην κατηγορία του πληρώματος καμπίνας ή αλλιώς των PNC ‘s (PersonnelNavigantCommercial). Στο πρόβλημά μας η ειδικότητα του κάθε ιπτάμενου ποσοτικοποιείται με μία ακέραια τιμή. Έτσι λοιπόν, για τους

κυβερνήτες η τιμή της θέσης είναι 1, για τους συγκυβερνήτες είναι 2, και για το προσωπικό καμπίνας είναι 3. Με τη λύση του προβλήματος καθένας θα πρέπει να έχει τοποθετηθεί στην αντίστοιχη θέση σε κάποια από τις διαθέσιμες εκπαιδευτικές συνεδρίες.

- **Λήξη (expiration)**. Κάθε αεροπορική εταιρεία είναι υποχρεωμένη να μεριμνά για τη συνεχή εκπαίδευση του προσωπικού της. Έτσι λοιπόν κάθε μέλος του ιπτάμενου προσωπικού πρέπει ανά τακτά χρονικά διαστήματα να παρακολουθεί κάποια μαθήματα έτσι ώστε να επιβεβαιώνονται οι ικανότητές του. Στο πρόβλημα που εξετάζεται,η λήξη της πτητικής ισχύος κάθε εργαζομένου εξαρτάται από τη θέση που κατέχει μέσα στο αεροπλάνο. Έτσι λοιπόν, κάθε πιλότος και συγκυβερνήτης (PNT ‘s)που εξετάζονται έχουν ένα περιθώριο τριών μηνών εντός του οποίου πρέπει να περάσουν από εκπαίδευση, ενώ το προσωπικό καμπίνας (PNC ‘s) πρέπει να περάσει από υποχρεωτική εκπαίδευση εντός του τρέχοντος μήνα. Η λήξη της πτητικής ικανότητας κάθε ιπταμένου αντιπροσωπεύεται από μια ακέραια τιμή. Η τιμή αυτή είναι 0 για όσους η πτητική ικανότητα λήγει τον τρέχοντα μήνα, 1 για εκείνους των οποίων λήγει τον επόμενο, και 2 για αυτούς που η πτητική ικανότητα λήγει τον μεθεπόμενο μήνα. Όπως αναφέρθηκε και παραπάνω η λήξη πτητικής ισχύος για το προσωπικό καμπίνας είναι πάντα 0.
- **Αρχαιότητα (Seniority)**. Κάθε μέλος του ιπτάμενου προσωπικού κατέχει επίσης ένα ακέραιο αριθμό ο οποίος καθορίζεται από την εκάστοτε αεροπορική εταιρεία ανάλογα με τα χρόνια προϋπηρεσίας και τις ώρες πτήσεις. Ο αριθμός αυτός είναι μοναδικός για κάθε εργαζόμενο, δηλαδή δεν γίνεται δύο ιπτάμενοι να έχουν τον ίδιο βαθμό αρχαιότητας. Όσο μεγαλύτερη είναι η τιμή αυτού του αριθμού τόσο μικρότερος είναι ο βαθμός αρχαιότητας.
- **Γλώσσα (Language)**. Κάθε άτομο που ανήκει στη λίστα των ιπταμένων διαθέτει επίσης ακόμη έναν ακέραιο αριθμό ο οποίος αντιπροσωπεύει την ευχέρεια κατανόησης και ομιλίας είτε αγγλικών είτε γαλλικών. Βέβαια μπορεί να υπάρξει περίπτωση ιπταμένων που γνωρίζουν και τις δύο γλώσσες . Έτσι λοιπόν, σε αυτούς που μιλούν γαλλικά δίνεται η τιμή 1 στο χαρακτηριστικό της γλώσσας, 2 δίνεται σε αυτούς που γνωρίζουν μόνο αγγλικά, και 0 σε όσους είναι εξοικειωμένοι και με τις δύο γλώσσες. Κάθε εκπαιδευτική συνεδρία πραγματοποιείται είτε στα αγγλικά είτε στα γαλλικά. Καταλαβαίνουμε λοιπόν πως είναι απαραίτητο να γνωρίζουμε ποια γλώσσα γνωρίζει ο κάθε ιπτάμενος έτσι ώστε να μην τοποθετήσουμε στην ίδια συνεδρία κάποιον που ξέρει μόνο γαλλικά με κάποιον που γνωρίζει μόνο αγγλικά.

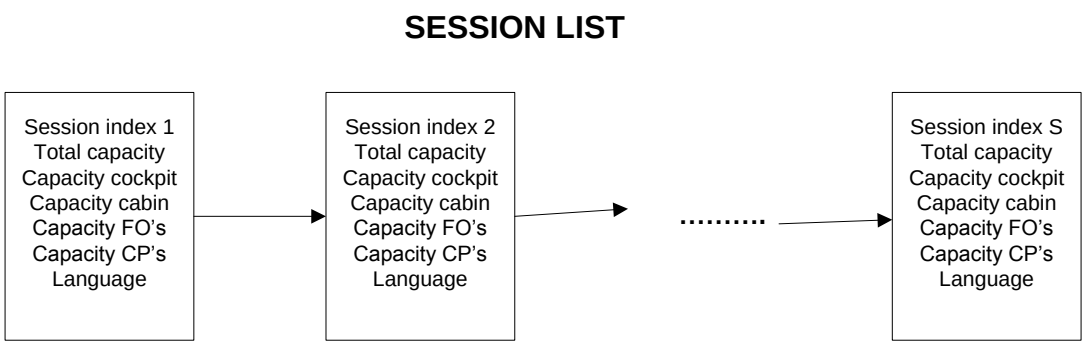
- **Λίστα εφικτών εκπαιδευτικών συνεδριών (List of valid training sessions)**.Για να μπορέσει κάποιος ιπτάμενος να συμμετάσχει σε μία εκπαιδευτική συνεδρία θα πρέπει να μην έχει άδεια και να μην έχει κάποια προγραμματισμένη πτήση. Κάθε ιπτάμενος, λοιπόν, έχει μία λίστα με τις εκπαιδευτικές συνεδρίες στις οποίες είναι εφικτή η ανάθεσή του και η οποία καθορίζεται με βάση τις ώρες τις οποίες είναι διαθέσιμος. Για κάθε συνεδρία που συμπεριλαμβάνεται στη λίστα αυτή, ο ιπτάμενος επιλέγει έναν ακέραιο αριθμό ο οποίος δηλώνει το βαθμό προτίμησης που έχει εκδηλώσει ο ιπτάμενος αυτός για τη συγκεκριμένη συνεδρία. Η τιμή 0 δηλώνει ότι ο συγκεκριμένος ιπτάμενος δεν έχει εκδηλώσει προτίμηση για τη συγκεκριμένη συνεδρία. Αντίθετα, μια θετική τιμή δηλώνει πως ο ιπτάμενος έχει εκφράσει προτίμηση για τη συγκεκριμένη συνεδρία. Σε αυτή την περίπτωση, όσο υψηλότερη είναι η θετική τιμή τόσο χαμηλότερη είναι η προτεραιότητα που έχει θέσει για αυτή τη συνεδρία ο ιπτάμενος. Μπορούν να υπάρξουν πολλές συνεδρίες της λίστας με μηδενική προτίμηση αλλά δεν μπορούν να υπάρξουν δύο ή περισσότερες με την ίδια θετική τιμή προτίμησης. Επιπλέον, οι θετικές τιμές προτεραιότητας θα πρέπει να ξεκινούν από 1 και να είναι συνεχόμενες χωρίς κενά (1, 2, 3..)



Εικόνα 3.1 : Σχηματική απεικόνιση της λίστας των ιπταμένων

Εκτός από τη λίστα των ιπτάμενων έχουμε και τη *λίστα με τις διαθέσιμες εκπαιδευτικές συνεδρίες του μήνα* που αποτελεί επίσης δεδομένο εισόδου του προβλήματος. Η κάθε συνεδρία διαθέτει δύο χαρακτηριστικά τα οποία επιβάλλουν περιορισμούς στη λειτουργία του συστήματος. Αυτά είναι η γλώσσα των λειτουργιών της συνεδρίας και η χωρητικότητα της αίθουσας όσον αφορά τον συνολικό αριθμό των εκπαιδευόμενων αλλά και της κάθε ειδικότητας ξεχωριστά. Αναλυτικότερα :

- **Χωρητικότητα (Capacity)**: Σε κάθε συνεδρία μπορεί να ανατεθεί ένα πλήθος Ν εκπαιδευόμενων. Μετά το τέλος της ανάλυσης του προβλήματος μπορεί σε μια ή περισσότερες συνεδρίες να έχουν ανατεθεί λιγότεροι από Ν ιπτάμενοι αλλά σε καμία περίπτωση ο αριθμός τους δεν μπορεί να είναι μεγαλύτερος από Ν. Επιπλέον καθορίζονται περιορισμοί χωρητικότητας που αφορούν τον μέγιστο αριθμό των εκπαιδευομένων στην καμπίνα αλλά και στο πιλοτήριο. Και τέλος γίνεται ακόμη ένας περαιτέρω διαχωρισμός που αφορά το μέγιστο αριθμό από κάθε ειδικότητα που μπορεί να ανατεθεί στη συγκεκριμένη συνεδρία. Οι χωρητικότητες αυτές επιβάλλουν περιορισμούς στη λειτουργία του συστήματος και δεν πρέπει να παραβιάζονται σε καμία περίπτωση.
- **Γλώσσα (Language)**: Η γλώσσα στη οποία πραγματοποιείται κάθε συνεδρία μπορεί να είναι είτε Γαλλικά (FR) είτε Αγγλικά (EN) ενώ υπάρχει και μια ακόμη επιλογή που δηλώνει ότι δεν έχει οριστεί κάποια γλώσσα για τη συγκεκριμένη συνεδρία. Στην περίπτωση αυτή η γλώσσα στην οποία πραγματοποιείται η εκάστοτε συνεδρία προκύπτει μετά τη λύση του προβλήματος. Στην περίπτωση όπου η γλώσσα της συνεδρίας ορίζεται από την αρχή απαραίτητη προϋπόθεση για να ανατεθεί ένας ιπτάμενος σε αυτή είναι να κατανοεί και να ομιλεί τη γλώσσα (εκτός βέβαια από το γεγονός ότι η συνεδρία πρέπει να ανήκει στη λίστα των εφικτών συνεδριών του).



Εικόνα 3.2: Σχηματική απεικόνιση των διαθέσιμων εκπαιδευτικών συνεδριών

3.2. Περιγραφή λύσης

Έπειτα από την καταγραφή των δεδομένων εισόδου κάνουμε αναφορά στα βασικά χαρακτηριστικά της λύσης του προβλήματος. Βασικός στόχος είναι η μέγιστη δυνατή αξιοποίηση των ανθρώπινων πόρων λαμβάνοντας υπόψη μας τους λειτουργικούς περιορισμούς του συστήματος.

Κάθε μέλος του πληρώματος με λήξη πτητικής ικανότητας τον τρέχοντα μήνα πρέπει να ανατεθεί σε μία από τις εφικτές συνεδρίες του. Οι υπόλοιπες άδειες θέσεις (αν υπάρχουν) καλύπτονται από ιπταμένους των οποίων η πτητική ικανότητα λήγει τον επόμενο ή μεθεπόμενο μήνα μέχρις ότου να μην υπάρχουν άλλες θέσεις ή να μην υπάρχουν άλλοι διαθέσιμοι ιπτάμενοι.

Οι περιορισμοί χωρητικότητας που αφορούν το σύνολο των εκπαιδευόμενων αλλά και κάθε ειδικότητα ξεχωριστά πρέπει να πληρούνται.

Κάθε εκπαιδευόμενος πρέπει να κατανοεί τη γλώσσα στην οποία πραγματοποιείται η συνεδρία στην οποία έχει ανατεθεί. Σε περίπτωση που η γλώσσα δεν καθορίζεται από την αρχή αλλά είναι αποτέλεσμα της λύσης, τότε όλοι όσοι έχουν ανατεθεί σε μία συνεδρία θα πρέπει να κατανοούν την ίδια γλώσσα.

Το πρόγραμμα προσπαθεί να ικανοποιήσει την πρώτη προτίμηση κάθε ιπταμένου τηρώντας τους περιορισμούς αυστηρής αρχαιότητας. Δεδομένου ότι κάθε ιπτάμενος με λήξη πτητικής ικανότητας 0 ανατίθεται σε μία εφικτή συνεδρία και ότι ο μέγιστος δυνατός αριθμός ιπταμένων ανατίθεται στις διαθέσιμες συνεδρίες, το πρόβλημα προσπαθεί να ικανοποιήσει την μέγιστη επιθυμία των αρχαιότερων ιπταμένων και ούτω καθεξής.

Για να ικανοποιηθεί η μέγιστη επιθυμία κάποιου συγκεκριμένου ιπτάμενου θα πρέπει να μην «διαταράσεται» καμία προτίμηση κανενός πιο αρχαίου ιπτάμενου, να μην αναιρείται η ανάθεση κάποιου με λήξη ισχύος 0 σε μία από τις εφικτές του εκπαιδευτικές συνεδρίες, και να μην αυξάνει ο αριθμός των άδειων θέσεων. Όταν λοιπον δεν μπορεί να γίνει αυτό, το πρόγραμμα προσπαθεί να ικανοποιήσει τη δεύτερη επιθυμία του, ή αν αυτό δεν είναι πάλι δυνατό, την τρίτη και ούτω καθεξής, χωρίς να μας ενδιαφέρει η επίδραση που θα έχει η ενέργεια αυτή στην ικανοποίηση της επιθυμίας ενός νεότερου ιπταμένου.

Εάν έπειτα από την ανάθεση όλων των ιπταμένων με λήξη 0 σε μία εκπαιδευτική συνεδρία και έπειτα από την μέγιστη δυνατή ικανοποίηση των επιθυμιών του προσωπικού υπάρχουν ακόμα άδειες θέσεις, τότε το πρόγραμμα αναθέτει τους υπόλοιπους ιπταμένους σε αυτές τις άδειες θέσεις χρησιμοποιώντας την ακόλουθη διαδικασία αντίστροφης αρχαιότητας. Σύμφωνα με αυτή τη διαδικασία, κάθε ιπτάμενος του οποίου η πτητική ικανότητα λήγει τον επόμενο μήνα πρέπει να

ανατεθεί σε μία από τις εφικτές του εκπαιδευτικές συνεδρίες για την οποία δεν είχε εκδηλώσει επιθυμία (τιμή προτεραιότητας = 0) πριν από οποιοδήποτε άλλο αρχαιότερο ιπτάμενο ίδιας θέσης ο οποίος επίσης έχει εφικτή τη συγκεκριμένη συνεδρία και δεν είχε εκδηλώσει επιθυμία, και πριν από οποιοδήποτε άλλο ιπτάμενο ίδιας θέσης με λήξη πτητικής ισχύος τον μεθεπόμενο μήνα ο οποίος και πάλι έχει εφικτή αυτή τη συνεδρία με τιμή προτεραιότητας 0. Ομοίως, ιπτάμενος με λήξη πτητικής ικανότητας τον μεθεπόμενο μήνα (expiration = 2) θα ανατεθεί σε μία από τις εφικτές του συνεδρίες με τιμή προτεραιότητας 0 πριν από οποιοδήποτε άλλο αρχαιότερο ιπτάμενο ίδιας θέσης με λήξη ισχύος 2 που έχει εφικτή τη συνεδρία χωρίς να έχει εκφράσει κάποια προτίμηση γι αυτή.

Το πρόγραμμα προσπαθεί να διατηρήσει μια ισορροπία (balance) ως προς τον αριθμό των εκπαιδευόμενων σε κάθε συνεδρία αλλά όχι εις βάρος των προτιμήσεών τους. Αυτό σημαίνει πως μία ισορροπημένη λύση του προβλήματος είναι προτιμότερη από μία λιγότερο ισορροπημένη σε περίπτωση που αυτή η επιλογή δεν επιδρά στην ικανοποίηση των προτιμήσεων των ιπταμένων. Όσον αφορά την ισορροπία του συστήματος, ο χρήστης του προγράμματος έχει δύο επιλογές. Η πρώτη επιλογή (heavybalance) προσπαθεί να διατηρήσει ισορροπία που αφορά όχι μόνο τον συνολικό αριθμό των ιπταμένων που ανατίθενται σε κάθε συνεδρία αλλά και τον αριθμό ιπταμένων ίδιας θέσης (κυβερνήτες, συγκυβερνήτες και προσωπικό καμπίνας) που συμμετέχουν σε αυτή. Με τη δεύτερη επιλογή (lightbalance) ο χρήστης προσπαθεί να διατηρήσει μια ισορροπία ως προς τον συνολικό αριθμό των ιπταμένων της κάθε συνεδρίας και τον αριθμό ιπταμένων του πιλοτηρίου και της καμπίνας. Σε ορισμένες περιπτώσεις, η πρώτη επιλογή μπορεί να οδηγήσει σε μεγάλους χρόνους επίλυσης του προβλήματος.

Εάν δεν υπάρχει εφικτή λύση με τα παραπάνω χαρακτηριστικά, το πρόγραμμα τερματίζει τη διαδικασία εμφανίζοντας στο χρήστη σχετικό μήνυμα (Theproblemisinfeasible).

4. Μαθηματική μοντελοποίηση του προβλήματος

Επόμενο βήμα για την επίλυση του προβλήματος είναι η μαθηματική μοντελοποίησή του, η οποία είναι απαραίτητη για τη δημιουργία του προγράμματος. Στο κεφάλαιο αυτό περιγράφονται τα δεδομένα εισόδου και δίνεται η μαθηματική παρουσίαση της αντικειμενικής συνάρτησης και των περιορισμών του προβλήματος.

4.1.Δεδομένα εισόδου

Για τη μαθηματική διατύπωση του προβλήματος, χρησιμοποιούμε τους εξής μαθηματικούς συμβολισμούς:

Σύνολα	
N	Σύνολο ιπταμένων (κυβερνήτες, συγκυβερνήτες και προσωπικό καμπίνας)
S	Σύνολο εκπαιδευτικών συνεδριών

Πίνακας 4.1 :Σύνολα του μαθηματικού μοντέλου

Δείκτες	
I	Ιπτάμενος (κυβερνήτης , συγκυβερνήτης ή προσωπικό καμπίνας)
j	Εκπαιδευτική συνεδρία

Πίνακας 4.2 :Δείκτες του μαθηματικού μοντέλου

Παράμετροι	
c_{ij}	Συντελεστής κόστους των μεταβλητώναπόφασηςx_{ij}της αντικειμενικής συνάρτησης
b_i	Συντελεστής κόστους των μεταβλητών απόφασηςy_iτης αντικειμενικής συνάρτησης
c_w	Συντελεστής κόστους της μεταβλητής απόφασης wτης αντικειμενικής συνάρτησης
p	Συντελεστής κόστους των μεταβλητώναπόφασης $v_{pos,j}$της αντικειμενικής συνάρτησηςπου είναι ίδιος για αυτές τις μεταβλητές
$position_i$	Θέση του ιπτάμενου (1 για κυβερνήτη, 2 για συγκυβερνήτη και 3 για πλήρωμα καμπίνας)
$expiration_i$	Λήξη ισχύος του ιπτάμενου i(0 αν λήγει τον τρέχοντα μήνα, 1 αν λήγει τον επόμενο και 2 τον μεθεπόμενο μήνα)
$seniority_i$	Αρχαιότητα ιπταμένου i (ακέραιος αριθμός, μοναδικός για κάθε ιπτάμενο ο οποίος όσο μεγαλύτερη τιμή παίρνει τόσο μικρότερη αρχαιότητα εκφράζει)
$language_i$	Γλώσσα που ομιλεί και κατανοεί ο iιπτάμενος (1 για Γαλλικά, 2 για Αγγλικά και 0 όταν γνωρίζει και τις δύο γλώσσες)
$priority_{ij}$	Τιμή προτεραιότητας του iιπτάμενου για τη jεφικτή συνεδρία (0 αν δεν έχει εκφράσει επιθυμία, 1 για τη μεγαλύτερη

	επιθυμία, 2 για τη δεύτερη και ούτω καθεξής)
MaxClassTrainees[j]	Χωρητικότητα της εκπαιδευτικής συνεδρίας j
MaxCK[j]	Χωρητικότητα για το πληρωμα πιλοτηρίου της εκπαιδευτικής συνεδρίας j
MaxCB[j]	Χωρητικότητα για το πληρωμα καμπίνας της εκπαιδευτικής συνεδρίας j
MaxClassCrew[pos][j]	Χωρητικότητα της εκπαιδευτικής συνεδρίας jγια ιπταμένους με θέση pos (όπου pos = 1 για πιλότους, pos = 2 για συγκυβερνήτες και pos = 3 για πλήρωμα καμπίνας)

Πίνακας 4.3 : Συμβολισμοί των παραμέτρων του μοντέλου

4.2.Μεταβλητές απόφασης

Οι μεταβλητές απόφασης που υπεισέρχονται στην αντικειμενική συνάρτηση είναι είτε δυαδικές είτε ακέραιες

Δυαδικές μεταβλητές απόφασης

Η πρώτη κατηγορία δυαδικών μεταβλητών απόφασης ορίζεται για την ανάθεση κάθε ιπταμένου i (που ανήκει στο σύνολο N) σε κάθε εκπαιδευτική συνεδρία j που ανήκει στη λίστα εφικτών συνεδριών του i ιπτάμενου. Έτσι έχουμε;

$$x_{ij}=\begin{cases}1\text{ , αν ανατεθεί ο ιπτάμενος i στην εκπαιδευτική συνεδρία j}\\ 0\text{, αν όχι}\end{cases}$$

Η δεύτερη κατηγορία ορίζεται για εκείνους τους ιπταμένους των οποίων η πτητική ικανότητα λήγει τον επόμενο ή μεθεπόμενο μήνα.

$$y_i=\begin{cases}1\text{ , αν ο ιπτάμενος i δεν ανατεθεί σε κάποια συνεδρία του τρέχοντος μήνα}\\ 0\text{, αν ανατεθεί}\end{cases}$$

Η τρίτη και τελευταία κατηγορία ορίζει την γλώσσα στην οποία θα πραγματοποιηθεί η κάθε συνεδρία

$$u_j=\begin{cases}1\text{, αν η γλώσσα της συνεδρίας j είναι τα Γαλλικά}\\ 0\text{, αν η γλώσσα της συνεδρίας j είναι τα Αγγλικά}\end{cases}$$

Οι μεταβλητές απόφασης που αφορούν την γλώσσα της συνεδρίας δεν προστίθενται στην αντικειμενική συνάρτηση.

Ακέραιες μεταβλητές απόφασης

Η πρώτη ακέραια μεταβλητή απόφασης δίνει τον αριθμό των ιπταμένων που δεν περνούν από εκπαίδευση.

$$w = ο\ συνολικός\ αριθμός\ των\ ιπταμένων\ που\ δεν\ ανατίθεται\ στις\ συνεδρίες$$

Επιπλέον, για κάθε ζεύγος εκπαιδευτικών συνεδριών και για κάθε θέση ξεχωριστά ορίζονται δύο ακέραιες μεταβλητές απόφασης ,οι $v_{pos,k}$και $v_{pos,j}$. Μόνο μία από τις δύο θα έχει θετική τιμή μετά την επίλυση του προβλήματος και η τιμή αυτή υποδηλώνει τη διαφορά στον αριθμό των ιπταμένων κάθε ειδικότητας που ανατίθενται στις συγκεκριμένες συνεδρίες.

4.3.Μαθηματικό μοντέλο

Έχοντας καταγράψει τα δεδομένα εισόδου και ορίσει τις μεταβλητές απόφασης επόμενο βήμα είναι η περιγραφή της αντικειμενικής συνάρτησης και του στόχου της καθώς και η περιγραφή των περιορισμών που την πλαισιώνουν.

Αντικειμενική συνάρτηση

Η μαθηματική μορφή της αντικειμενικής συνάρτησης είναι η εξής:

$$\boldsymbol{min} \sum_{i=1}^{N}\sum_{j=1}^{S} \boldsymbol{c_{ij}x_{ij}} + \sum_{i=1}^{N} \boldsymbol{b_i\, y_i} + \boldsymbol{c_w\, w} + (\boldsymbol{S-1})\boldsymbol{p} \sum_{pos=0}^{m}\sum_{j=1}^{S} \boldsymbol{v_{pos,j}}$$

Ο πρώτος όρος αθροισμάτων αφορά τις μεταβλητές ανάθεσης του iιπτάμενου στη j συνεδρία. Ο δεύτερος όρος αναφέρεται στις μεταβλητές μη ανάθεσης για εκπαίδευση τον τρέχοντα μήνα. Ο τρίτος όρος αναφέρεται στην ακέραια μεταβλητή απόφασης w. Και ο τελευταίος όρος αθροισμάτων αφορά τις ακέραιες μεταβλητές ισορροπίας. Σε περίπτωση που ο χρήστης επιλέξει lightbalance για το σύστημα η παράμετρος m παίρνει την τιμή 2, ενώ αν επιλέξει heavybalance θα πάρει την τιμή 3.

Στόχος της αντικειμενικής συνάρτησης είναι πρωτίστως να μειώσει τον αριθμό των ιπταμένων που δεν προορίζονται για εκπαίδευση αλλά και να μειώσει το κόστος ανάθεσης των ιπταμένων στις συνεδρίες το οποίο συνεπάγεται την μέγιστη δυνατή ικανοποίηση των ιπταμένων τηρώντας σειρά αυστηρής αρχαιότητας. Αν κάποια συνεδρία j δεν ανήκει στη λίστα των εφικτών συνεδριών του ιπτάμενου i τότε δεν

ορίζεται η μεταβλητή x_{ij}. Ομοίως αν κάποιος ιπτάμενος i έχει λήξη ισχύος τον τρέχοντα μήνα τότε δεν ορίζεται και η μεταβλητή y_i.

Ο τρόπος με τον οποίο προκύπτουν οι συντελεστές κόστους που είναι και οι συντελεστές των μεταβλητών απόφασης στην αντικειμενική συνάρτηση δίνεται στη συνέχεια. Πάντως πρέπει να αναφερθεί ότι ο συντελεστής κόστους που αντιστοιχεί στη συνεδρία j για την οποία ο ιπτάμενος i έχει εκδηλώσει τη μέγιστη επιθυμία (τιμή προτεραιότητας = 1) είναι μηδέν. Ενώ για τις υπόλοιπες εφικτές συνεδρίες του οι συντελεστές κόστους αυξάνονται όσο αυξάνονται οι τιμές προτεραιότητας με τέτοιο τρόπο ώστε να δίνεται προτεραιότητα στην ικανοποίηση ιπταμένων με μεγαλύτερο βαθμό αρχαιότητας. Δεν πρέπει να ξεχνάμε ότι πρωταρχικός στόχος του προβλήματος είναι να ανατεθούν όσο το δυνατόν περισσότεροι ιπτάμενοι για εκπαίδευση και για αυτό το λόγο ο συντελεστής κόστους της μεταβλητής w παίρνει και μεγαλύτερη τιμή από όλους τους άλλους συντελεστές.

Περιορισμοί

- Περιορισμοί ανάθεσης

Για κάθε ιπτάμενο itου οποίου η πτητική ικανότητα λήγει τον τρέχοντα μήνα ισχύει:

$$\boldsymbol{x_{i1} + x_{i2} + \ ... + x_{ij} = 1}$$

Οι όροι του αθροίσματος αφορούν μόνο τις εφικτές συνεδρίες του ιπτάμενου. Με τον περιορισμό αυτό εξασφαλίζεται ότι ο ιπτάμενος με λήξη ισχύος 0 θα ανατεθεί για εκπαίδευση τον τρέχοντα μήνα.

Για κάθε ιπτάμενο i του οποίου η πτητική ικανότητα λήγει τον επόμενο ή τον μεθεπόμενο μήνα ισχύει:

$$\boldsymbol{x_{i1} + x_{i2} + \ ... + x_{ij} + y_i = 1}$$

Στην περίπτωση αυτή αν y_i =0 τότε ο ιπτάμενος i πρέπει να ανατεθεί σε μία από τις εφικτές του συνεδρίες ενώ αν y_i = 1 τότε δε θα περάσει από εκπαίδευση αυτό τον μήνα.

- Περιορισμοί χωρητικότητας

Για κάθε εκπαιδευτική συνεδρία j ισχύει ο περιορισμός:

$$\boldsymbol{x_{1j} + x_{2j} + \ ... + x_{nj} \ \leq \ MaxClassTrainees[j]}$$

Οι όροι του αθροίσματος στο πρώτο μέλος περιλαμβάνουν τους ιπτάμενους που έχουν εφικτή τη συνεδρία j. Με τον περιορισμό αυτό εξασφαλίζεται ότι ο αριθμός των εκπαιδευόμενων που ανατίθενται στη συγκεκριμένη συνεδρία δεν υπερβαίνει τη μέγιστη χωρητικότητα της τάξης.
Επίσης για κάθε εκπαιδευτική συνεδρία ισχύουν παρόμοιοι περιορισμοί που αφορούν τον μέγιστο αριθμό ιπτάμενων πιλοτηρίου και καμπίνας.

$$\boldsymbol{x_{1j} + x_{2j} + \ \ldots + x_{nj} \ \leq \ MaxCK[j]}$$

Οι όροι του αριστερού μέλους αφορούν τους κυβερνήτες και συγκυβερνήτες που έχουν εφικτή τη συνεδρία j. Το άθροισμα τους δεν θα πρέπει σε καμία περίπτωση να υπερβαίνει τον διαθέσιμο αριθμό θέσεων της συνεδρίας που προορίζονται αποκλειστικά για το προσωπικό του πιλοτηρίου.
Ομοίως για το πλήρωμα καμπίνας ισχύει

$$\boldsymbol{x_{1j} + x_{2j} + \ \ldots + x_{nj} \ \leq \ MaxCB[j]}$$

Οι όροι του αριστερού μέλους αφορούν τοπροσωπικό καμπίνας που έχει εφικτή τη συνεδρία j. Το άθροισμα τους δεν θα πρέπει σε καμία περίπτωση να υπερβαίνει τον διαθέσιμο αριθμό θέσεων που προορίζονται αποκλειστικά για αυτή την κατηγορία.
Προχωρώντας την ανάλυση ένα βήμα παραπέρα, δημιουργούμε περιορισμούς χωρητικότητας που αφορούν κάθε θέση (position) ξεχωριστά

$$\sum_{i \in N: position_i = pos} x_{ij} \leq \ MaxClassCrew[pos][j]$$

Δημιουργούμε έτσι τρεις ομάδες περιορισμών. Μία για τους κυβερνήτες (pos = 1), μια για τους συγκυβερνήτες (pos = 2) και μία για το πλήρωμα καμπίνας (pos = 3). Με τους περιορισμούς αυτούς εξασφαλίζουμε ότι δε θα παραβιαστεί η χωρητικότητα κάθε εκπαιδευτικής συνεδρίας jγια κάθε μία από τις τρεις θέσεις.

- Περιορισμοί αντίστροφης αρχαιότητας για τον μήνα X+1

Για κάθε ιπτάμενο iτου οποίου η πτητική ικανότητα λήγει τον επόμενο μήνα και για κάθε εφικτή του συνεδρία jμε τιμή προτεραιότητας 0 ισχύει ο περιορισμός:

$$x_{i1} + x_{i2} + \ ... + x_{ij} \ \geq \ \frac{\sum_{k \in K} x_{kj}}{|K|}$$

Το αριστερό μέλος του περιορισμού περιλαμβάνει εκτός από την j και όλες τις άλλες εφικτές συνεδρίες του i ιπτάμενου. Το σύνολο K στο δεξιό μέλος είναι το πλήθος των αρχαιότερων ιπταμένων που έχουν την ίδια θέση με τον i ιπτάμενο, έχουν εφικτή την συνεδρία j με τιμή προτεραιότητας 0 και η πτητική τους ικανότητα λήγει τον επόμενο μήνα, καθώς και των ιπτάμενων ίδιας θέσης με τον i, με λήξη ισχύος τον μεθεπόμενο μήνα που δεν έχουν εκφράσει καμία επιθυμία για τη συνεδρία j.
Με τον περιορισμό αυτό εξασφαλίζεται ότι κάθε ιπτάμενος του οποίου η πτητική ικανότητα λήγει τον επόμενο μήνα θα ανατεθεί σε μία συνεδρία για την οποία δεν έχει εκφράσει επιθυμία πριν από οποιοδήποτε άλλο αρχαιότερο ιπτάμενο ίδιας θέσης που επίσης δεν έχει εκφράσει επιθυμία για τη συγκεκριμένη συνεδρία και έχει λήξη ισχύος ίση με 1 αλλά και πριν από οποιοδήποτε άλλο ιπτάμενο ίδιας θέσης που η πτητική του ικανότητα λήγει τον μεθεπόμενο μήνα και έχει δηλώσει τιμή προτεραιότητας 0 για αυτή τη συνεδρία. Στο αριστερό μέλος μπαίνουν και οι υπόλοιπες εφικτές συνεδρίες του κάθε ιπτάμενου έτσι ώστε ο περιορισμός να είναι ανενεργός σε περίπτωση που ήδη έχει ανατεθεί σε μία από αυτές.
Επίσης για κάθε εκπαιδευόμενο i με λήξη πτητικής ισχύος τον επόμενο μήνα και για κάθε εφικτή του συνεδρία jμε τιμή προτεραιότητας 0 ισχύουν οι περιορισμοί:

$$z + x_{i1} + x_{i2} + \ ... + x_{ij} \ \geq \ \frac{\sum_{k \in K} x_{kj}}{|K|}$$

$$z \ \leq \ \frac{\sum_{t \in T} x_{tj}}{MaxClassCrewPos[r]}$$

Το αριστερό μέλος του πρώτου περιορισμού περιέχει επίσης όλες τις εφικτές συνεδρίες του ιπτάμενου i. Ενώ το σύνολο K εμπεριέχει όλους τους αρχαιότερους ιπταμένους διαφορετικής θέσης από τον εξεταζόμενο ιπτάμενο, με λήξη ισχύος τον επόμενο μήνα οι οποίοι δεν έχουν εκφράσει προτίμηση για τη j συνεδρία, καθώς και όλους τους ιπτάμενους διαφορετικής θέσης με τον i των οποίων η πτητική ικανότητα λήγει τον μεθεπόμενο μήνα και οι οποίοι επίσης δεν έχουν εκδηλώσει επιθυμία για την συγκεκριμένη συνεδρία. Το σύνολο T περιλαμβάνει όλους τους ιπταμένους ίδιας θέσης με τον i

- Των οποίων η πτητική ικανότητα λήγει τον τρέχοντα μήνα και έχουν εφικτή τη συνεδρία j

- Των οποίων η πτητική ικανότητα λήγει τον επόμενο μήνα, έχουν εφικτή τη συνεδρία j και οι οποίοι είτε έχουν μικρότερο βαθμό αρχαιότητας είτε έχουν εκδηλώσει επιθυμία για την j συνεδρία
- Και εκείνοι των οποίων η πτητική ικανότητα λήγει τον μεθεπόμενο μήνα, και έχουν εκδηλώσει επιθυμία για την j συνεδρία

Η *μεταβλητή z*είναι μια δυαδική μεταβλητή που παίρνει την τιμή 0 αν $\sum_{t\in T} \boldsymbol{x_{tj}}$<$\boldsymbol{MaxClassCrewPos}[\boldsymbol{r}]$, δηλαδή αν υπάρχουν άδειες θέσεις στην συνεδρία j για αυτή τη θέση και την τιμή 1 αν η συνεδρία είναι πλήρης για αυτή τη θέση. Όταν ισχύει η δεύτερη περίπτωση (όπου οι θέσεις είναι γεμάτες) η μεταβλητή zμπορεί να πάρει και την τιμή 0 στο δεύτερο περιορισμό. Αλλά αν κατά την επίλυση του προβλήματος θέλουμε να αναθέσουμε κάποιους από τους ιπτάμενους i στη j συνεδρία τότε η μεταβλητή z θα πάρει αναγκαστικά την τιμή 1 καθώς λόγω της πλήρωσης των θέσεων δεν είναι δυνατόν να ανατεθεί ο ιπτάμενος i σε αυτή τη συνεδρία. Η μεταβλητή z δεν υπεισέρχεται στην αντικειμενική συνάρτηση.

Η παράμετρος *r* δηλώνει τη θέση του ιπτάμενου i και η παράμετρος *MaxClassCrewPos[r]* δηλώνει την χωρητικότητα της j συνεδρίας για το προσωπικό με θέση r.

Αν το σύνολο T είναι άδειο τότε η μεταβλητή z και ο δεύτερος περιορισμός δεν προστίθενται στο μαθηματικό μοντέλο.

- Περιορισμοί αντίστροφης αρχαιότητας για τον μήνα X+2

Για κάθε ιπτάμενο iτου οποίου η πτητική ικανότητα λήγει τον μεθεπόμενο μήνα και για κάθε εφικτή συνεδρία jμε τιμή προτεραιότητας 0 ισχύει ο περιορισμός:

$$\boldsymbol{x_{i1}} + \boldsymbol{x_{i2}} + \ \ldots + \boldsymbol{x_{ij}} \ \geq \ \frac{\sum_{\boldsymbol{k}\in \boldsymbol{K}} \boldsymbol{x_{kj}}}{|\boldsymbol{K}|}$$

Το αριστερό μέλος του περιορισμού περιλαμβάνει εκτός από την j και όλες τις άλλες εφικτές συνεδρίες του i ιπτάμενου. Το σύνολο K στο δεξιό μέλος είναι το πλήθος των αρχαιότερων ιπταμένων που έχουν την ίδια θέση με τον iιπτάμενο, έχουν εφικτή την συνεδρία j με τιμή προτεραιότητας 0 και η πτητική τους ικανότητα λήγει τον μεθεπόμενο μήνα..

Με τον περιορισμό αυτό εξασφαλίζεται ότι κάθε ιπτάμενος του οποίου η πτητική ικανότητα λήγει τον μεθεπόμενο μήνα θα ανατεθεί σε μία συνεδρία για την οποία δεν έχει εκφράσει επιθυμία πριν από οποιοδήποτε άλλο αρχαιότερο ιπτάμενο ίδιας θέσης που επίσης δεν έχει εκφράσει επιθυμία για τη συγκεκριμένη συνεδρία και έχει λήξη ισχύος ίση με 2 μήνες. Στο αριστερό μέλος μπαίνουν και οι υπόλοιπες εφικτές

συνεδρίες του κάθε ιπτάμενου έτσι ώστε ο περιορισμός να είναι ανενεργός σε περίπτωση που ήδη έχει ανατεθεί σε μία από αυτές.
Επίσης για κάθε εκπαιδευόμενο i με λήξη πτητικής ισχύος τον μεθεπόμενο μήνα και για κάθε εφικτή του συνεδρία j με τιμή προτεραιότητας 0 ισχύει :

$$\boldsymbol{z + x_{i1} + x_{i2} + \ \ldots + x_{ij} \ \geq \ \frac{\sum_{k \in K} x_{kj}}{|K|}}$$

$$\boldsymbol{z \ \leq \ \frac{\sum_{t \in T} x_{tj}}{MaxClassCrewPos[r]}}$$

Το αριστερό μέλος του πρώτου περιορισμού περιέχει επίσης όλες τις εφικτές συνεδρίες του ιπτάμενου i. Ενώ το σύνολο K εμπεριέχει όλους τους αρχαιότερους ιπταμένους διαφορετικής θέσης από τον εξεταζόμενο ιπτάμενο, με λήξη ισχύος τον μεθεπόμενο μήνα οι οποίοι δεν έχουν εκφράσει προτίμηση για τη jσυνεδρία. Το σύνολο Τ περιλαμβάνει όλους τους ιπταμένους ίδιας θέσης με τον i

- Των οποίων η πτητική ικανότητα λήγει τον τρέχοντα μήνα ή τον μεθεπόμενο και έχουν εφικτή τη συνεδρία j
- Των οποίων η πτητική ικανότητα λήγει τον μεθεπόμενο μήνα, έχουν εφικτή τη συνεδρία j και οι οποίοι είτε είναι νεότεροι είτε έχουν εκδηλώσει επιθυμία για την j συνεδρία

Η *μεταβλητή z* είναι μια δυαδική μεταβλητή που παίρνει την τιμή 0 αν $\sum_{t \in T} x_{tj} < MaxClassCrewPos[r]$, δηλαδή αν υπάρχουν άδειες θέσεις στην συνεδρία j για αυτή τη θέση και την τιμή 1 αν η συνεδρία είναι πλήρης για αυτή τη θέση. Όταν ισχύει η δεύτερη περίπτωση (όπου οι θέσεις είναι γεμάτες) η μεταβλητή zμπορεί να πάρει και την τιμή 0 στο δεύτερο περιορισμό. Αλλά αν κατά την επίλυση του προβλήματος θέλουμε να αναθέσουμε κάποιους από τους ιπτάμενους i στη j συνεδρία τότε η μεταβλητή z θα πάρει αναγκαστικά την τιμή 1 καθώς λόγω της πλήρωσης των θέσεων δεν είναι δυνατόν να ανατεθεί ο ιπτάμενος iσε αυτή τη συνεδρία. Η μεταβλητή z δεν υπεισέρχεται στην αντικειμενική συνάρτηση.
Η παράμετρος *r* δηλώνει τη θέση του ιπτάμενου iκαι *MaxClassCrewPos[r]*δηλώνει την χωρητικότητα της j συνεδρίας για το προσωπικό με θέση r.
Αν το σύνολο Τ είναι άδειο τότε η μεταβλητή zκαι ο δεύτερος περιορισμός δεν προστίθενται.

- Περιορισμός μη κενών θέσεων

$$\sum_{i \in N} y_i = w$$

Το σύνολο N περιλαμβάνει όλους τους ιπταμένους των οποίων η άδεια πτήσης δεν λήγει τον τρέχοντα μήνα και για τους οποίους έχουν οριστεί οι αντίστοιχες μεταβλητές y_i. Η μεταβλητή w είναι μια ακέραια μεταβλητή απόφασης. Με τον περιορισμό αυτό υπολογίζουμε τον αριθμό των ιπταμένων που δεν θα πραγματοποιήσουν εκπαίδευση αυτό τον μήνα.

- Περιορισμοί ισορροπίας

Για κάθε θέση pos(position) και για κάθε ζεύγος εκπαιδευτικών συνεδριών jκαι kπροσθέτουμε τον ακόλουθο περιορισμό:

$$\sum_{l \in L} x_{lj} + v_{lj} = \sum_{r \in R} x_{rk} + v_{rk}$$

Το σύνολο L περιέχει όλο το προσωπικό ίδιας θέσης pos που έχουν εφικτή τη j συνεδρία και το σύνολο R περιέχει όλο το προσωπικό ίδιας θέσης pos που έχουν εφικτή τη k συνεδρία.Προσπαθούμε όσο είναι δυνατό να αναθέτουμε σε κάθε συνεδρία τον ίδιο αριθμό εκπαιδευόμενων γενικά αλλά και για κάθε θέση ειδικότερα.Οι μεταβλητές v_{lj} και v_{rk} είναι δύο ακέραιες μεταβλητές απόφασης (μοναδικές για κάθε ζεύγος συνεδριών και για κάθε θέση) οι οποίες συμμετέχουν στην αντικειμενική συνάρτηση με ένα μικρό συντελεστή κόστους που είναι ίδιος για όλες. Το πολύ μία από αυτές τις δύο μεταβλητής θα πάρει θετική τιμή ενώ η άλλη θα είναι μηδέν. Αυτή η θετική τιμή εκφράζει τη διαφορά στον αριθμό των ιπταμένων μεταξύ των συνεδριών j και k για κάθε θέση ξεχωριστά. Με αυτό τον περιορισμό,το πρόβλημα προσπαθεί να βρει εκείνη τη λύση με την οποία ο αριθμός των ιπταμένων διαφέρει ελάχιστα έως και καθόλου από συνεδρία σε συνεδρία γενικά αλλά και για κάθε θέση ξεχωριστά με τέτοιο τρόπο ώστε να μην επηρεάζεται η προτίμηση των ιπταμένων. Όπως αναφέρθηκε ήδη ο χρήστης του προγράμματος έχει δύο επιλογές. Η πρώτη επιλογή (heavybalance) προσπαθεί να εξισορροπήσει όχι μόνο τον συνολικό αριθμό των ιπταμένων που ανατίθενται σε κάθε συνεδρία αλλά και τον αριθμό ιπταμένων ίδιας θέσης (κυβερνήτες, συγκυβερνήτες και προσωπικό καμπίνας) που συμμετέχουν σε κάθε συνεδρία. Με τη δεύτερη επιλογή (lightbalance) ο χρήστης προσπαθεί να διατηρήσει ισορροπία ως προς τον συνολικό αριθμό των ιπταμένων της κάθε συνεδρίας και τον αριθμό ιπταμένων του πιλοτηρίου και της καμπίνας

- Περιορισμοί γλώσσας

Για κάθε εκπαιδευτική συνεδρία jγια την οποία δεν έχει αποφασιστεί εξαρχής η γλώσσα ισχύει:

$$\frac{\sum_{r \in R} x_{rj}}{|R|} \leq u_j$$

$$\frac{\sum_{n \in N} x_{nj}}{|N|} \leq 1 - u_j$$

Το σύνολο R περιέχει όλους τους ιπτάμενους μιλούν μόνο γαλλικά και η συνεδρία j περιλαμβάνεται στη λίστα εφικτών συνεδριών τους ενώ το σύνολο N περιέχει εκείνους που μιλούν μόνο αγγλικά και έχουν εφικτή τη συνεδρία j. Η μεταβλητή u_j είναι μία δυαδική μεταβλητή απόφασης η οποία δεν υπεισέρχεται στην αντικειμενική συνάρτηση. Με τους περιορισμούς αυτούς εξασφαλίζουμε τη συμβατότητα κάθε ιπταμένου με τη γλώσσα της συνεδρίας στην οποία και θα ανατεθεί. Έτσι όταν $u_j = 1$ η γλώσσα ανάθεσης στη συνεδρία j είναι τα γαλλικά και όταν $u_j = 0$ η γλώσσα ανάθεσης είναι τα αγγλικά. Υπάρχει βέβαια η περίπτωση να έχει οριστεί εξαρχής η γλώσσα της συνεδρίας οπότε οι παραπάνω περιορισμοί δεν επιτρέπουν την ανάθεση στη συνεδρία αυτή κάποιο ιπταμένου που δεν γνωρίζει τη συγκεκριμένη γλώσσα.

4.4. Προσδιορισμός των συντελεστών κόστους των μεταβλητών απόφασης

Όπως αναφέρθηκε και σε προηγούμενη ενότητα κάθε ιπτάμενος έχει μία λίστα με τις εκπαιδευτικές συνεδρίες στις οποίες μπορεί να ανατεθεί. Για κάθε μία από αυτές τις συνεδρίες, ο ιπτάμενος θα πρέπει να εκφράσει την προτίμησή του με έναν ακέραιο αριθμό. Η συνεδρία με τιμή προτίμησης 1 είναι αυτή στην οποία ο ιπτάμενος θα ήθελε να ανατεθεί. Η αμέσως επόμενη προτίμηση του ορίζεται με τον αριθμό 2 και ούτω καθεξής. Όσο μεγαλύτερος είναι ο θετικός ακέραιος αριθμός μιας συνεδρίας τόσο μικρότερη είναι και η προτίμηση του ιπτάμενου για τη συνεδρία αυτή επομένως τόσο μεγαλύτερη θα είναι η δυσαρέσκεια του αν ανατεθεί σε αυτή. Επίσης υπάρχει η περίπτωση ο ιπτάμενος να μην εκφράσει επιθυμία για μία ή περισσότερες συνεδρίες. Οι συνεδρίες αυτές θα έχουν τιμή προτεραιότητας μηδέν. Επίσης η ανάθεση του σε μία από αυτές τις συνεδρίες του δημιουργεί μεγαλύτερη δυσαρέσκεια από το να ανατεθεί σε μία συνεδρία για την οποία είχε εκφράσει επιθυμία.

Δημιουργούμε μία λίστα η οποία ονομάζεται PenaltyList και αποτελείται από κόμβους καθένας από τους οποίους αντιπροσωπεύει από μία εφικτή συνεδρία κάθε ιπτάμενου. Οι κόμβοι αυτοί αναφέρονται στις μεταβλητές απόφασης x_{ij}. Επίσης δημιουργούνται κόμβοι μη ανάθεσης για όσους η πτητική ικανότητα λήγει τον

επόμενο ή μεθεπόμενο μήναοι οποίοι και αναφέρονται στις μεταβλητές y_i. Τέλος, υπάρχει ένας κόμβος που αντιστοιχεί στη μεταβλητή w, η οποία δηλώνει τον συνολικό αριθμό που δεν περνά από εκπαίδευση αυτό το μήνα. Στη συνέχεια ταξινομούμε τους κόμβους της λίστας κατάλληλα έτσι ώστε να μπορέσουμε να δώσουμε τιμές στους συντελεστές κόστους της αντικειμενικής συνάρτησης.

Στην PenaltyList,οι κόμβοι που αφορούν τον ίδιο ιπτάμενο γειτνιάζουν ενώ προς το τέλος της λίστας τοποθετούνται οι αρχαιότεροι ιπτάμενοι. Τελευταίος κόμβος της λίστας αντιστοιχεί πάντα στη μεταβλητή w. Για τον κάθε ιπτάμενο ξεχωριστά πρώτα τοποθετείται η εφικτή συνεδρία με τη μικρότερη θετική τιμή και ακολουθούν οι υπόλοιπες με αύξουσα τιμή προτεραιότητας. Έπειτα τοποθετείται ο κόμβος μη ανάθεσης (αν υπάρχει) και τέλος τοποθετούνται οι συνεδρίες με τιμή προτεραιότητας 0 σε τυχαία σειρά.

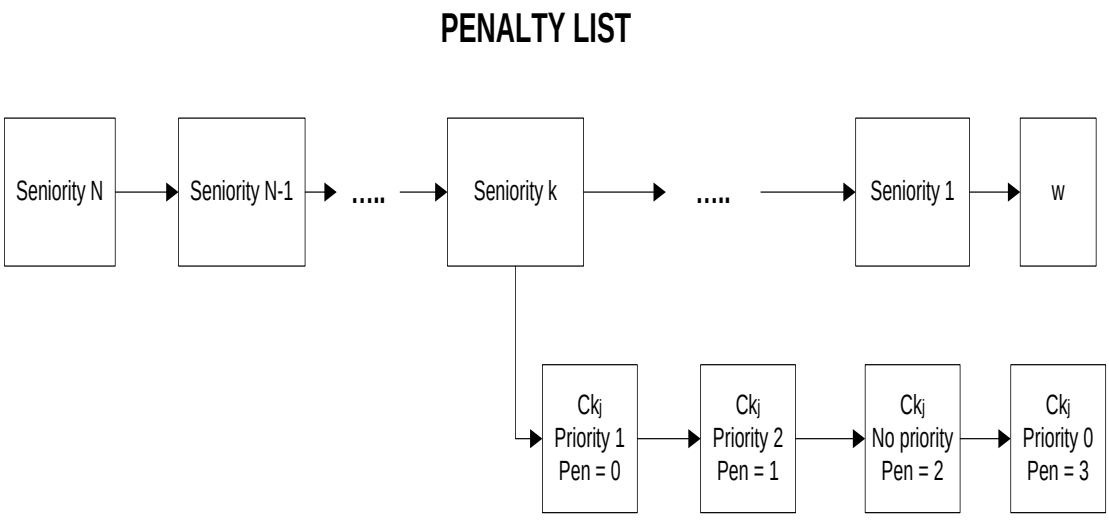
Έχοντας περιγράψει τη διάταξη αυτής της λίστας, δίνεται στη συνέχεια ο τρόπος με τον οποίο αυτό επιτυγχάνεται στην πράξη. Δημιουργούμε ένα διάνυσμα *max_priority (i)* το οποίο περιέχει τη μέγιστη τιμή προτεραιότητας των ιπτάμενων. Έπειτα δημιουργούμε μια βοηθητική μεταβλητή την οποία ονομάζουμε *prior* και την οποία εισάγουμε σε κάθε κόμβο της λίστας. Για τις εφικτές συνεδρίες για τις οποίες δεν έχει εκφράσει επιθυμία ο ιπτάμενος, ισχύει prior =0.Αν ο ιπτάμενος ιέχει δηλώσει θετική προτεραιότητα k για μία συνεδρία jτότε η μεταβλητή prior ισούται με max_priority(i) – k + 2. Για όσους ιπτάμενους η πτητική τους ικανότητα λήγει τον επόμενο μήνα στον αντίστοιχο κόμβο η μεταβλητή prior παίρνει την τιμή 1. Για τον κόμβο που αντιστοιχεί στη μεταβλητή w δεν χρειάζεται να οριστεί η μεταβλητή prior.

Έτσι λοιπόν ο κόμβος της μεταβλητής w τοποθετείται στο τέλος της λίστας. Οι κόμβοι των αρχαιότερων ιπταμένων τοποθετούνται προς το τέλος της λίστας με φθίνουσα σειρά της παραμέτρου seniority. Ενώ οι κόμβοι του ίδιου ιπταμένου τοποθετούνται με φθίνουσα σειρά της τιμής prior. Έχοντας ταξινομήσει τη λίστα προχωράμε στον τρόπο προσδιορισμού των συντελεστών κόστους.

Για κάθε ιπτάμενο ο συντελεστής κόστους για την συνεδρία με τη μεγαλύτερη τιμή prior(μεγαλύτερη επιθυμία) παίρνει την τιμή 0. Η συνεδρία με την αμέσως μικρότερη τιμή prior έχει μεγαλύτερο συντελεστή κόστους. Οι συντελεστές κόστους για τις συνεδρίες χωρίς επιθυμία (prior = 0) είναι ακόμα μεγαλύτεροι από τους προηγούμενους αλλά ίσοι μεταξύ τους. Η λογική ανάθεσης τιμών για τον αμέσως αρχαιότερο ιπτάμενο είναι η ίδια με μία διαφορά. Ο πρώτος θετικός συντελεστής κόστους (εκτός από αυτόν που αντιστοιχεί στη συνεδρία με τη μέγιστη επιθυμία που είναι πάντα ίσος με 0) πρέπει να είναι μεγαλύτερος από το άθροισμα όλων των μεγαλύτερων συντελεστών των ιπταμένων με μικρότερο βαθμό αρχαιότητας, ενώ κάθε επόμενος συντελεστής του ίδιου ιπταμένου θα πρέπει να αυξάνει τουλάχιστον

κατά το άθροισμα αυτό σε σχέση με το αμέσως προηγούμενο. Με τον τρόπο αυτό εξασφαλίζεται ότι θα ικανοποιηθεί η μέγιστη επιθυμία κάθε ιπταμένου έστω και αν αυτό γίνεται εις βάρος των νεότερων συναδέλφων του.

Η μεταβλητή w έχει συντελεστή κόστους μεγαλύτερο από το άθροισμα των μέγιστων συντελεστών κόστους όλων των ιπτάμενων και αυτό γιατί πρωταρχικός στόχος είναι να μειώσουμε το σύνολο των ιπτάμενων που δεν ανατίθενται σε συνεδρία των τρέχοντα μήνα.



Εικόνα 4.1 : Σχηματική απεικόνιση της PenaltyList

Σε περίπτωση όπου το πλήθος των ιπτάμενων και των εκπαιδευτικών συνεδριών είναι μεγάλο καταλαβαίνουμε πως οι συντελεστές κόστους μπορούν να φτάσουν σε αστρονομικές τιμές πράγμα που μπορεί να δημιουργήσει προβλήματα στην διαδικασία επίλυσης του προβλήματος.

Για να αποφύγουμε κάτι τέτοιο λύνουμε το πρόβλημα σε στάδια. Ορίζουμε τους συντελεστές κόστους για τους κόμβους συγκεκριμένου αριθμού ιπταμένων ξεκινώντας από το τέλος της λίστας προς την αρχή και στους υπόλοιπους αναθέτουμε συντελεστές κόστους ίσους με 0. Λύνουμε το πρόβλημα και κλειδώνουμε τις αναθέσεις των ιπταμένων. Επαναλαμβάνουμε αυτή τη διαδικασία μέχρις ότου εξαντλήσουμε όλους τους κόμβους.

Αυτό που πρέπει να προσέξουμε είναι να ορίσουμε συντελεστές κόστους σε όλους τους κόμβους που σχετίζονται με έναν ιπτάμενο και όχι μόνο σε μερικούς. Για το λόγο αυτό ο αριθμός των συντελεστών κόστους που θα πάρουν τιμές προσαρμόζονται κατάλληλα ώστε να συμπεριληφθούν όλοι οι συντελεστές κόστους του τελευταίου από αριστερά ιπταμένου που θα είναι ενεργός στο συγκεκριμένο στάδιο της επίλυσης.

Σε κάθε επανάληψη της διαδικασίας για όσες μεταβλητές,μετά την επίλυση, έχουν πάρει τιμή 1 και αντιστοιχούν σε ανάθεση που δεν έχει προτίμηση 0 για έναν ιπτάμενο τις κλειδώνουμε εισάγοντας στο μοντέλο έναν περιορισμό που επιβάλλει την τιμή αυτή. Οι περιορισμοί αυτοί παραμένουν ενσωματωμένοι στο μοντέλο σε όλες τις επαναλήψεις. Οι μεταβλητές απόφασης που αντιστοιχούν στη χαμηλότερη επιθυμία ενός ιπταμένου δεν κλειδώνονται ποτέ έστω και αν πάρουν την τιμή 1, επειδή αντιπροσωπεύουν ούτως ή άλλως τη χαμηλότερη ικανοποίηση που μπορεί να έχει ο αντίστοιχος ιπτάμενος. Έτσι δίνονται ακόμα περισσότερες επιλογές συνεδριών για τους νεότερους ιπτάμενους χωρίς αυτό να γίνεται εις βάρος των αρχαιότερων.

5. Περιγραφή του κώδικα σε γλώσσα C

Για την ανάπτυξη του κώδικα χρησιμοποιούμε τη γλώσσα προγραμματισμού C και για τη βελτιστοποίηση χρησιμοποιούμε το λογισμικό πακέτο βελτιστοποίησης IBMILOGCPLEXOptimizationStudio. Η σύνθεση του κώδικα πραγματοποιήθηκε στο προγραμματιστικό περιβάλλον MicrosoftVisualC++ Express 2010. Ο κώδικας περιλαμβάνεται στο Παράρτημα Α.

Όταν εκτελείται κώδικας που έχει γραφτεί στη γλώσσα προγραμματισμού C εκτελείται πρώτα το κομμάτι του κώδικα που βρίσκεται εντός της συνάρτησης main(). Σε περιπτώσεις όπου ο κώδικας είναι αρκετά μεγάλος και πολύπλοκος, κομμάτια του κώδικα που επιτελούν συγκεκριμένο έργο αποτελούν αυτόνομες συναρτήσεις οι οποίες καλούνται μέσα από τη main(). Επίσης, μία συνάρτηση που έχει κληθεί μέσα από τη main() μπορεί να καλεί με τη σειρά της μία άλλη συνάρτηση κ.ο.κ. Μετά από κάθε τέτοια κλήση, η ροή του προγράμματος επιστρέφει στην αμέσως επόμενη γραμμή από αυτή της κλήσης της συνάρτησης. Συνηθίζεται μάλιστα (όπως και στο συγκεκριμένο κώδικα) η συνάρτηση main() να βρίσκεται στο τέλος του προγράμματος, χωρίς να αλλάζει κάτι στη λογική του προγράμματος.

Στον κώδικα που δημιουργούμε,καλούνται οι εξής συναρτήσεις από τη συνάρτηση main ():

1. init_problem ()
2. fil_lists ()
3. print_lists ()
4. clock ()
5. add_columns()
6. add_assignments()
7. add_capacity_trainees()
8. add_capacity_cockpit_cabin()

9. add_capacity_per_position()
10. no_empty_seats()
11. add_language()
12. Fix_Reverse_X1()
13. Fix_Reverse_diffID_X1()
14. Fix_Reverse_X2()
15. Fix_Reverse_diffID_X2()
16. balance(Balancelevel)
17. DoPenalty(PList)
18. solve_evaluate_results()
19. clock()
20. free_problem()

Στη συνέχεια παρουσιάζεται κάθε μία συνάρτηση ξεχωριστά. Επίσης περιγράφουμε και μία ακόμα συνάρτηση η οποία δεν καλείται από τη main() αλλά από την συνάρτηση add_columns() και η οποία ονομάζεται AddtoPenaltyList.

1. *Η συνάρτηση init_problem:*

Η συνάρτηση αυτή περιλαμβάνει δύο ρουτίνες: την *CPXopenCPLEX* και την *CPXcreateprob*. Η πρώτη χρησιμοποιείται για να διαμορφώσουμε το περιβάλλον της CPLEX στο οποίο θα αναπτύξουμε τον κώδικα. Η ρουτίνα αυτή αφού εκτελεστεί επιστρέφει ένα δείκτη (*env*) ο οποίος δείχνει στο περιβάλλον που δημιουργήθηκε.Η δεύτερη ρουτίνα χρησιμοποιείται για να δημιουργήσουμε ένα πρόβλημα της CPLEX μέσα στο περιβάλλον που δημιουργήσαμε με την πρώτη ρουτίνα. Η ρουτίνα αυτή επιστρέφει ένα δείκτη (*lp*) ο οποίος δείχνει στο πρόβλημα που μόλις δημιουργήθηκε. Το πρόβλημα που δημιουργήθηκε είναι ένα πρόγραμμα ελαχιστοποίησης με μηδενικό αριθμό μεταβλητών απόφασης και μηδενικό αριθμό περιορισμών

2. *Η συνάρτηση fill_lists*

Με αυτή τη συνάρτηση εισάγουμε στον κώδικα τα δεδομένα εισόδου του προβλήματος. Αυτά είναι:

- Η λίστα των ιπταμένων που ονομάζεται *PilotList.* Το κάθε μέλος της λίστας διαθέτει τα εξής χαρακτηριστικά:
    - Αρχαιότητα (seniority) που περιγράφεται από έναν ακέραιο αριθμό, μοναδικό για κάθε ιπτάμενο. Για τον αρχαιότερο ιπτάμενο ισχύει

seniority = 1 και για όλους τους υπόλοιπους η αρχαιότητα αυξάνεται κατά μία μονάδα. Έτσι λοιπόν ο νεότερος ιπτάμενος έχει seniority = N.

- Θέση (position) με τιμή 1 για τους κυβερνήτες, 2 για συγκυβερνήτες και 3 για το πλήρωμα καμπίνας και πάλι η θέση κάθε ιπτάμενου προκύπτει από μια γεννήτρια τυχαίων αριθμών.
- Λήξη πτητικής ισχύος (expiration). Η λήξη ισχύος κάθε ιπταμένου μπορεί να είναι 0, 1 ,2 με εξαίρεση αυτή του πληρώματος καμπίνας που είναι πάντα 0. Η λήξη ισχύος των πιλότων και των συγκυβερνητών καθορίζεται από γεννήτρια τυχαίων αριθμών.
- Γλώσσα (language) που παίρνει την τιμή 0 αν μιλά και αγγλικά και γαλλικά, 1 αν μιλά μόνο γαλλικά και 2 αν μιλά μόνο αγγλικά. Και πάλι η τιμή της γλώσσας προκύπτει με τη χρήση γεννήτριας τυχαίων αριθμών.
- Λίστα με τις εφικτές εκπαιδευτικές συνεδρίες (listofvalidbids) και την καταγραφή των τιμών προτεραιότητας αυτών (priority).

- Λίστα με τις εκπαιδευτικές συνεδρίες (*SessionList)* όπου κάθε στοιχείο της έχεις τα εξής χαρακτηριστικά:
  - Συνολική χωρητικότητα (totalcapacity). Για το αρχικό πρόβλημα θεωρούμε πως όλες οι θέσεις των συνεδριών είναι κενές και δεν έχει γίνει κάποια προ-ανάθεση ιπτάμενου.
  - Χωρητικότητα καμπίνας (capacityincabin) που είναι ένας ακέραιος αριθμός που πρέπει να είναι μικρότερος από την συνολική χωρητικότητα της εκάστοτε συνεδρίας
  - Χωρητικότητα πιλοτηρίου (capacityincockpit)που είναι ένας ακέραιος αριθμός που πρέπει να είναι μικρότερος από την συνολική χωρητικότητα της εκάστοτε συνεδρίας
  - Χωρητικότητα πιλότων (capacityCP) που είναι ένας ακέραιος αριθμός μικρότερος ή ίσος από την χωρητικότητα πιλοτηρίου.
  - Χωρητικότητα συγκυβερνητών (capacityFO) που είναι ένας ακέραιος αριθμός μικρότερος από την χωρητικότητα πιλοτηρίου.
  - Γλώσσα (language) με τιμή 0 διότι θεωρούμε πως η γλώσσα της κάθε συνεδρίας δεν είναι καθορισμένη από την αρχή του προβλήματος.

3. *Η συνάρτηση print_lists*

Με την συνάρτηση αυτή τυπώνουμε σε ένα αρχείο στο οποίο δίνουμε το όνομα *file* τα χαρακτηριστικά κάθε ιπτάμενου και κάθε εκπαιδευτικής συνεδρίας. Για κάθε ιπτάμενο της PilotList τυπώνουμε:

- Έναν αύξοντα αριθμό(pilotindex)
- Λήξη πτητικής ισχύος(expiration)
- Θέση (position)
- Αρχαιότητα (seniority)
- Γλώσσα (language)
- Σύνολο εφικτών εκπαιδευτικών συνεδριών (num_ofvalidbids)
- Τις τιμές προτεραιότητας των συνεδριών που ανήκουν σε αυτή τη λίστα

Για κάθε διαθέσιμη συνεδρία του τρέχοντος μήνα τυπώνουμε στο ίδιο αρχείο

- Αύξων αριθμός της συνεδρίας (index)
- Γλώσσα (language)
- Συνολική χωρητικότητα (total_capacity)
- Χωρητικότητα πιλοτηρίου (capacity_in_cockpit)
- Χωρητικότητα καμπίνας (capacity_in_cabin)
- Χωρητικότητα πιλότων (capacity_CP)
- Χωρητικότητα συγκυβερνητών (capacity_FO)

4. *Η συνάρτηση clock()*

Ορίζεται το σημείο εκκίνησης της καταγραφής του χρόνου επίλυσης του προβλήματος.

5. *Η συνάρτηση add_columns*

Με αυτή την συνάρτηση εισάγονται στον κώδικα οι μεταβλητές απόφασης που αφορούν την ανάθεση των ιπταμένων στις εκπαιδευτικές συνεδρίες (μεταβλητές x_{ij}) αλλά και αυτές που αντιπροσωπεύουν την μη ανάθεση τους σε κάποια συνεδρία αυτού του μήνα (μεταβλητές y_i).

Αρχικά για κάθε μια μεταβλητή ξεχωριστά δεσμεύουμε μία θέση μνήμης χρησιμοποιώντας τη συνάρτηση *getmem_newcols*. Έπειτα σαρώνουμε την PilotList και για κάθε ιπτάμενο αυτής της λίστας σαρώνουμε τη λίστα των διαθέσιμων συνεδριών (SessionList) για να βρούμε τις εφικτές του συνεδρίες. Για κάθε εφικτή συνεδρία που βρίσκουμε εισάγουμε την αντίστοιχη μεταβλητή απόφασης καλώντας την ρουτίνα *CPXnewcols*. Με αυτή τη ρουτίνα καθορίζουμε τον συντελεστή της μεταβλητής που αρχικά τον θέτουμε 0. Επίσης καθορίζονται τα όρια των τιμών που μπορεί να πάρει η μεταβλητή απόφασης μετά τη λύση του προβλήματος. Στο δικό

μας πρόβλημα το άνω όριο των μεταβλητών x_{ij} είναι 1 και το κάτω όριο είναι 0.Καθορίζεται, επίσης, ο τύπος της μεταβλητής (π.χ αν είναι δυαδική όπως στη δική μας περίπτωση) καθώς και το όνομα της το οποίο δεν είναι απαραίτητο να ορισθεί. Σε περίπτωση που η ρουτίνα καταφέρει να εισάγει την μεταβλητή επιστρέφει την τιμή 0. Διαφορετικά επιστρέφει τιμή διάφορη του μηδενός.

Αυτό που κάνουμε στη συνέχεια είναι να ορίσουμε έναν αύξοντα αριθμό σε κάθε μεταβλητή ο οποίος θα είναι και ο δείκτης που θα έχει μέσα στην αντικειμενική συνάρτηση. Αυτό το πετυχαίνουμε καλώντας τη ρουτίνα *CPXgetnumcols* η οποία μας δίνει πρόσβαση στον αριθμό των μεταβλητών που έχουν εισαχθεί στο πρόβλημα.

Να τονίσουμε εδώ πως κάθε μεταβλητή απόφασης δεν διαθέτει δύο δείκτες όπως στο μαθηματικό μοντέλο που αναπτύξαμε αλλά μόνο έναν που είναι μοναδικός και συμβολίζει την ανάθεση του iιπτάμενου στη j συνεδρία. Έπειτα κάθε μεταβλητή αφού εισαχθεί στο πρόβλημα εισάγεται και στην PenaltyList καλώντας την συνάρτηση *AddtoPenaltyList.*

Αν ο ιπτάμενος iέχει δηλώσει τιμή προτεραιότητας 0 για την εφικτή εκπαιδευτική συνεδρία jτότε στην PenaltyList εισάγεται ένας κόμβος με τα εξής χαρακτηριστικά

- RecType ‘P’ (για ανάθεση ή μη)
- Sen αρχαιότητα του ιπτάμενου i
- Prior 0
- IlogIdx αύξων αριθμός μεταβλητής στο cplex

Αν ο ιπτάμενος iέχει δηλώσει τιμή προτεραιότητας k>0 για την εφικτή εκπαιδευτική συνεδρία jτότε στην PenaltyListεισάγεται ένας κόμβος με τα εξής χαρακτηριστικά

- RecType ‘P’ (για ανάθεση ή μη)
- Sen αρχαιότητα του ιπτάμενου i
- Prior max_priority(i)-k+2
- IlogIdx αύξων αριθμός μεταβλητής στο cplex

Με τον ίδιο τρόπο εισάγονται στο πρόβλημα και στην PenaltyList και οι μεταβλητές ανάθεσης y_i. Οι μεταβλητές αυτές ορίζονται μόνο για τους ιπταμένους με λήξη πτητικής ισχύος τον επόμενο και μεθεπόμενο μήνα. Σαρώνουμε και πάλι την PilotList και για όσους έχουν expiration>0 εισάγουμε την μεταβλητή y_i. Μετά εισάγουμε και στην PenaltyList έναν κόμβο με τα εξής χαρακτηριστικά:

- RecType ‘P’ (για ανάθεση ή μη)
- Sen αρχαιότητα του ιπτάμενου i
- Prior 1
- IlogIdx αύξων αριθμός μεταβλητής στο cplex

Στο τέλος τηςσυνάρτησης ελευθερώνουμε τη μνήμη που είχαμε δεσμεύσει για τις μεταβλητές χρησιμοποιώντας την συνάρτηση *freemem_newcols*.

5.α. Η συνάρτηση AddtoPenaltyList

Με αυτή τη συνάρτηση εισάγουμε στην PenaltyListτους κόμβους που αντιστοιχούν στις μεταβλητές xij και yi και τους ταξινομούμε κατά τέτοιο τρόπο ώστε:

- ο κόμβος που αντιστοιχεί στη μεταβλητή w να πηγαίνει στο τέλος της λίστας
- οι κόμβοι με μεγαλύτερο seniority να τοποθετούνται στην αρχή της λίστας.
- Για την ίδια τιμή αρχαιότητας, δηλαδή για τον ίδιο ιπτάμενο, οι κόμβοι που αντιστοιχούν σε μεγαλύτερη τιμή της βοηθητικής μεταβλητής Prior να τοποθετούνται προς την αρχή της λίστας.

6. Η συνάρτηση add_assignments

Εισάγει περιορισμούς ανάθεσης κάθε ιπτάμενου. Αρχικά για κάθε ένα περιορισμό που αντιστοιχεί σε κάθε ιπτάμενο δεσμεύουμε την απαιτούμενη μνήμη με την συνάρτηση *getmem_addrows*. Έπειτα ορίζουμε δύο βοηθητικές μεταβλητές: την *cnt* με την οποία μετράμε τον αριθμό των μεταβλητών x_{ij} που ελέγχουμε και την *not_exp_cnt* με την οποία μετράμε τον αριθμό των μεταβλητών y_iπου έχουμε ελέγξει. Στην αρχή και οι δύο μεταβλητές ισούνται με 0.

Σαρώνουμε την PiloLlist και για κάθε στοιχείο της σαρώνουμε την SessionList προκειμένου να βρούμε τις εφικτές συνεδρίες κάθε ιπτάμενου. Για κάθε εφικτή συνεδρία που βρίσκουμε προσθέτουμε στο αριστερό μέρος του περιορισμού την αντίστοιχη μεταβλητή με δείκτη *cnt* και συντελεστή 1 και αυξάνουμε το *cnt* κατά ένα και προχωράμε στην επόμενη συνεδρία. Έχοντας προσθέσει όλες τις μεταβλητές που αντιστοιχούν στις εφικτές συνεδρίες ελέγχουμε αν ο ιπτάμενος έχει seniority>0. Αν συμβαίνει αυτό εισάγουμε στον περιορισμό και την μεταβλητή ανάθεσης με δείκτη που είναι ίσος με το άθροισμα του συνολικού αριθμού των εφικτών συνεδριών όλων των ιπταμένων και της μεταβλητής *not_exp_cnt* και συντελεστή 1 και αυξάνουμε την *not_exp_cnt* κατά 1.

Η εισαγωγή του περιορισμού γίνεται με την ρουτίνα *CPXaddrows* η οποία έχει ως παραμέτρους:

- Τον δείκτη σε περιβάλλον cplex
- Τον δείκτη σε πρόβλημα cplex
- Το σύνολο των περιορισμών που προστίθενται
- Το σύνολο των μεταβλητών στο αριστερό μέλος του περιορισμού
- Η διεύθυνση μνήμης της μεταβλητής που περιέχει την τιμή του δεξιού μέλος του περιορισμού. Στη δική μας περίπτωση το δεξιό μέλος είναι μονάδα
- Ο αριθμητικός τελεστής. Οι περιορισμοί ανάθεσης είναι περιορισμοί ισότητας.

- Και το όνομα του περιορισμού.

Στο τέλος ελευθερώνουμε τη μνήμη που έχουμε δεσμεύσει για τους περιορισμούς.

7. *Ησυνάρτηση add_capacity_trainees*

Με την υπορουτίνα αυτή εισάγουμε περιορισμούς που εξασφαλίζουν ότι ο συνολικός αριθμός των εκπαιδευόμενων που θα ανατεθεί σε κάθε συνεδρία δε θα υπερβαίνει το διαθέσιμο αριθμό θέσεων στη συνεδρία αυτή

Όπως και στην προηγούμενη υπορουτίνα ξεκινάμε δεσμεύοντας την απαιτούμενη μνήμη για τον κάθε περιορισμό. Σαρώνουμε την SessionList και για κάθε διαθέσιμη συνεδρία ελέγχουμε ποιοι πιλότοι την έχουν δηλώσει ως εφικτή και εισάγουμε την αντίστοιχη μεταβλητή στον περιορισμό. Έπειτα εισάγουμε τον περιορισμό με τη ρουτίνα *CPXaddrows,* που περιγράψαμε παραπάνω. Η διαδικασία αυτή επαναλαμβάνεται για όλες τις συνεδρίες και η ρουτίνα ολοκληρώνεται ελευθερώνοντας τη μνήμη που είχαμε δεσμεύσει για τους περιορισμούς.

8. *Ησυνάρτηση add_capacity_cockpit_cabin*

Εδώ εισάγονται οι περιορισμοί χωρητικότητας που εξασφαλίζουν ότι σε κάθε συνεδρία δε θα τοποθετηθούν περισσότεροι ιπτάμενοι καμπίνας και πιλοτηρίου από τις θέσεις που διατίθενται σε κάθε συνεδρία για κάθε κατηγορία ξεχωριστά. Η διαδικασία που ακολουθούμε είναι η εξής: ορίζουμε την απαιτούμενη μνήμη. Εκτελούμε δύο φορές ένα βρόγχο επανάληψης. Ο βρόγχος αυτός εκτελείται την πρώτη φορά για το προσωπικό του πιλοτηρίου (times = 1) και τη δεύτερη φορά για το πλήρωμα καμπίνας (times =2).Εντοπίζουμε και προσμετράμε κατά την πρώτη επανάληψη για κάθε συνεδρία τους εκπαιδευόμενους που έχουν θέση κυβερνήτη ή συγκυβερνήτη και που έχουν εφικτή την εκάστοτε συνεδρία και καταγράφουμε τις αντίστοιχες μεταβλητές ανάθεσης. Εισάγουμε για κάθε συνεδρία τον αντίστοιχο περιορισμό. Επαναλαμβάνουμε τη διαδικασία για κάθε συνεδρία εντοπίζοντας τους ιπτάμενους που ανήκουν στο πλήρωμα καμπίνας.

9. *Ησυνάρτηση add_capacity_per_position*

Με αυτή τη συνάρτηση εισάγονται οι περιορισμοί χωρητικότητας που εξασφαλίζουν ότι σε κάθε συνεδρία δε θα τοποθετηθούν περισσότεροι κυβερνήτες και συγκυβερνήτες από τις θέσεις που διατίθενται σε κάθε συνεδρία για κάθε θέση ξεχωριστά. Η διαδικασία που ακολουθούμε είναι παρόμοια με εκείνη της υπορουτίνας add_*capacity_cockpit_cabin*. Αρχικά ορίζουμε την απαιτούμενη μνήμη. Εκτελούμε δύο φορές ένα βρόγχο επανάληψης. Ο βρόγχος αυτός εκτελείται την

πρώτη φορά για τους κυβερνήτες (times = 1) και τη δεύτερη φορά για τους συγκυβερνήτες (times =2). Εντοπίζουμε και προσμετράμε κατά την πρώτη επανάληψη για κάθε συνεδρία τους εκπαιδευόμενους που έχουν θέση κυβερνήτη και που έχουν εφικτή την εκάστοτε συνεδρία και καταγράφουμε τις αντίστοιχες μεταβλητές ανάθεσης. Εισάγουμε για κάθε συνεδρία τον αντίστοιχο περιορισμό. Επαναλαμβάνουμε τη διαδικασία για κάθε συνεδρία εντοπίζοντας τους ιπτάμενους που ανήκουν στη θέση του συγκυβερνήτη.

10. Η συνάρτηση no_empty_seats

Εισάγεται η μεταβλητή w που αντιπροσωπεύει τον αριθμό των ιπτάμενων που δεν ανατίθενται για εκπαίδευση τον τρέχοντα μήνα.

Το πρώτο βήμα είναι να εισάγουμε τη μεταβλητή w χρησιμοποιώντας την ρουτίνα *CPXnewcols* προσέχοντας όμως να δηλώσουμε την νέα μεταβλητή ως ακέραια και όχι ως δυαδική όπως έγινε με την περίπτωση των υπόλοιπων μεταβλητών απόφασης.

Έπειτα, σαρώνουμε την λίστα με τους εκπαιδευόμενους προκειμένου να εντοπίσουμε εκείνους που έχουν expiration μεγαλύτερο του 0. Προσθέτουμε τις μεταβλητές απόφασης y_iπου αντιστοιχούν σε αυτούς τους εκπαιδευόμενους σε ένα περιορισμό που θέτει τη μεταβλητή w ίση με το άθροισμά τους. Έπειτα δημιουργούμε ένα κόμβο για τη μεταβλητή w και τον προσθέτουμε στην PenaltyList.καλώντας τη συνάρτηση AddtoPenaltyList. Τα χαρακτηριστικά του κόμβου αυτού είναι:

- RecTypeM (δηλώνει το κόστος της μεταβλητής w)
- Sen -1
- Prior -1
- IlogIdx αύξων αριθμός της μεταβλητής wστο cplex

Αυτή είναι η μοναδική μεταβλητή απόφασης με RecType ‘M’ στην PenaltyList και ο συντελεστής κόστους της στην αντικειμενική συνάρτηση είναι μεγαλύτερος από οποιοδήποτε άλλο συντελεστή κόστους. Η μεγάλη τιμή κόστους της αναγκάζει το πρόγραμμα να αναζητήσει την λύση εκείνη που πάνω από όλα ελαχιστοποιεί τον αριθμό των πιλότων που δεν ανατίθενται σε εκπαιδευτική συνεδρία αυτού του μήνα.

11. Η συνάρτηση add_language

Με την συνάρτηση αυτή εξασφαλίζεται ότι η γλώσσα που ομιλεί κάθε ιπτάμενος θα είναι συμβατή με τη γλώσσα της συνεδρίας στην οποία και θα ανατεθεί. Επίσης βρίσκει και μια εφικτή γλώσσα ανάθεσης για τις συνεδρίες για τις οποίες δεν έχει

προκαθοριστεί η γλώσσα. Η διαδικασία που ακολουθείται αναλύεται στα παρακάτω βήματα:

- Ορίζουμε την απαιτούμενη μνήμη στο πρόγραμμα για να καταχωρίσουμε στη συνέχεια τις δυαδικές μεταβλητές απόφασης u_j που ορίζουν την γλώσσα ανάθεσης για τη συνεδρία j. Επίσης ορίζουμε την απαιτούμενη μνήμη για τους περιορισμούς ανάθεσης της γλώσσας.
- Σαρώνουμε τη λίστα με τις διαθέσιμες συνεδρίες αυτού του μήνα και για κάθε μία για την οποία δεν έχει προκαθοριστεί η γλώσσα εκτελούμε δύο φορές ένα βρόγχο επανάληψης , μία για τα γαλλικά και μια για τα αγγλικά. Με αυτή τον βρόγχο καταμετράμε εκείνους τους ιπταμένους που έχουν εφικτή τη συνεδρία j και των οποίων η γλώσσα είναι ίδια με εκείνη που ελέγχεται κάθε φορά από το βρόχο επανάληψης.
- Βρίσκουμε τις μεταβλητές απόφασης που τους αντιστοιχούν και τις εισάγουμε στον περιορισμό
- Επαναλαμβάνουμε τη διαδικασία για κάθε συνεδρία της SessionList
- Ελευθερώνουμε τις θέσεις μνήμης που είχαμε δεσμεύσει στην αρχή της συνάρτησης.

12. Η συνάρτηση Fix_Reverse_X1()

Εκτελούμε ένα βρόγχο επανάληψης δύο φορές. Την πρώτη φορά για τους κυβερνήτες και τη δεύτερη για τους συγκυβερνήτες. Επισημαίνουμε πως το πλήρωμα καμπίνας πρέπει οπωσδήποτε να ανατεθεί για εκπαίδευση αυτό το μήνα και για αυτό το λόγο δεν εκτελούμε τον βρόγχο για τη δική τους περίπτωση. Σε κάθε επανάληψη λοιπόν σαρώνεται η λίστα των εκπαιδευόμενων και αναζητούνται εκείνοι των οποίων η άδεια πτήσης λήγει τον επόμενο μήνα και των οποίων η θέση είναι όμοια με εκείνη για την οποία εκτελείται ο βρόγχος επανάληψης. Για κάθε έναν από αυτούς, σαρώνουμε τη λίστα με τις εφικτές του συνεδρίες και αναζητούμε εκείνες για τις οποίες ο ιπτάμενος δεν έχει δηλώσει προτίμηση (priority = 0). Για καθεμία από αυτές τις συνεδρίες καλούμε τη συνάρτηση *find_NoOfSeniorIDs*.

Η συνάρτηση *find_NoOfSeniorIDs* παίρνει ως ορίσματα τον ιπτάμενο και τη συνεδρία που εξετάζεται και βρίσκει το πλήθος των ιπταμένων οι οποίοι έχουν ίδια θέση με τον υπό εξέταση ιπτάμενο, έχουν εφικτή την συγκεκριμένη συνεδρία με τιμή προτεραιότητας 0 και είτε είναι αρχαιότεροι από τον εξεταζόμενο ιπτάμενο με λήξη πτητικής ισχύος 1 είτε έχουν λήξη πτητικής ισχύος 2.

Αφού η συνάρτηση *find_NoOfSeniorIDs* βρει το πλήθος αυτών των ιπταμένων επιστρέφει την τιμή αυτή στη συνάρτηση *Fix_Reverse_X1()*. Σε περίπτωση που ο

αριθμός αυτός είναι θετικός προστίθεται στο πρόβλημα ένας ακόμη περιορισμός ακολουθώντας την εξής διαδικασία:
Σαρώνουμε πάλι την PilotList και αναζητούμε ξανά τους ιπταμένους εκείνους το πλήθος των οποίων βρήκαμε με την συνάρτηση *find_NoOfSeniorIDs*. Για κάθε έναν από αυτούς αναζητούμε στη λίστα των εφικτών συνεδριών του τη συνεδρία εκείνη που είναι ίδια με την εξεταζόμενη συνεδρία και που έχει μηδενική προτεραιότητα. Στη συνέχεια εισάγουμε στον περιορισμό την μεταβλητή που αντιστοιχεί σε αυτό τον πιλότο και αυτή τη συνεδρία. Έχοντας τελειώσει με αυτούς τους ιπτάμενους προσθέτουμε στον περιορισμό και τις μεταβλητές απόφασης που αντιστοιχούν στον εξεταζόμενο ιπτάμενο.
Η διαδικασία αυτή όπως είπαμε επαναλαμβάνεται δύο φορές. Μία για τη θέση του κυβερνήτη και μία για εκείνη του συγκυβερνήτη.

13. ΗσυνάρτησηFix_Reverse_diffID_X1()

Η διαδικασία που ακολουθείται σε αυτή τη συνάρτηση είναι παρόμοια με εκείνη της συνάρτησης *Fix_Reverse_X1()*, μόνο που τώρα για κάθε ιπτάμενο και για κάθε εφικτή του συνεδρία με προτεραιότητα 0 ίσως χρειαστεί να προστεθεί και μία επιπλέον δυαδική μεταβλητή, η *μεταβλητή z* την οποία είχαμε αναφέρει στο κεφάλαιο 4.3. Πιο συγκεκριμένα:

- Οι εντολές της συνάρτησης περιλαμβάνονται σε ένα βρόγχο επανάληψης που εκτελείται δύο φορές, μία για τη θέση του κυβερνήτη και μία για τη θέση του συγκυβερνήτη.
- Σαρώνουμε τη λίστα των ιπταμένων αναζητώντας εκείνους με θέση ίδια με τη θέση για την οποία εκτελείται ο βρόγχος επανάληψης και με λήξη πτητικής ισχύος τον επόμενο μήνα.
- Για κάθε έναν ελέγχουμε τη λίστα των εφικτών συνεδριών του αναζητώντας εκείνες με τιμή προτεραιότητας μηδέν.
- Για κάθε μία από αυτές καλούμε τη συνάρτηση *NoOfdiffSeniorIDs*. Η συνάρτηση αυτή υπολογίζει το πλήθος των ιπταμένων που έχουν τα ακόλουθα χαρακτηριστικά: δεν ανήκουν στην ίδια θέση με τον εξεταζόμενο πιλότο, έχουν εφικτή την εξεταζόμενη συνεδρία με μηδενική τιμή προτεραιότητας και είτε η πτητική τους ισχύς λήγει τον επόμενο μήνα και είναι αρχαιότεροι από τον εξεταζόμενο ιπτάμενο είτε η πτητική τους ισχύς λήγει τον μεθεπόμενο μήνα. Η τιμή που υπολογίζει η συνάρτηση *NoOfdiffSeniorIDs* επιστρέφεται στην συνάρτηση από την οποία και κλήθηκε.

- Αν η τιμή που επιστράφηκε είναι θετική προστίθεται τουλάχιστον ένας ακόμη περιορισμός που αφορά τον εξεταζόμενο ιπτάμενο και την εξεταζόμενη συνεδρία του. Σε διαφορετική περίπτωση προχωράμε στη επόμενη εφικτή συνεδρία με προτεραιότητα μηδέν κ.ο.κ
- Στην περίπτωση λοιπόν που η τιμή που επιστρέφεται είναι θετική υπολογίζεται στη συνέχεια το σύνολο των πιλότων που έχουν ίδια θέση με τον εξεταζόμενο ιπτάμενο και:
    - η πτητική τους ισχύς λήγει τον τρέχοντα μήνα και έχουν εφικτή την υπό εξέταση συνεδρία
    - η πτητική τους ισχύς λήγει τον επόμενο μήνα και είτε είναι νεότεροι από τον ιπτάμενος μας έχοντας εφικτή την υπό εξέταση συνεδρία είτε έχουν δηλώσει προτίμηση για αυτή
    - η πτητική τους ισχύς λήγει τον μεθεπόμενο μήνα και έχουν εκφράσει προτίμηση για την υπό εξέταση συνεδρία

  Το πλήθος του συνόλου αυτού το αναπαριστούμε με μία μεταβλητή που ονομάζουμε *TSET*
- Εφόσον το σύνολο αυτό δεν είναι κενό προστίθεται εκτός από τον περιορισμό αντίστροφης αρχαιότητας και ένας ακόμη περιορισμός που ελέγχει αν για την εξεταζόμενη συνεδρία έχουν πληρωθεί όλες οι θέσεις που αφορούν τη θέση που εξετάζεται κάθε φορά από τον βρόγχο επανάληψης. Επίσης προστίθεται και η μεταβλητή z. Σε περίπτωση που το σύνολο είναι κενό ο δεύτερος περιορισμός και η μεταβλητή z δεν προστίθενται.
- Για να προσθέσουμε τον περιορισμό αντίστροφης αρχαιότητας του εξεταζόμενου πιλότου και συνεδρίας σαρώνουμε ξανά την PilotPist και αναζητούμε ξανά τους ιπταμένους εκείνους το πλήθος των οποίων βρήκαμε με την συνάρτηση *NoOfdiffSeniorIDs*. Για κάθε έναν από αυτούς αναζητούμε στη λίστα των εφικτών συνεδριών του τη συνεδρία εκείνη που είναι ίδια με την εξεταζόμενη συνεδρία και που έχει μηδενική προτεραιότητα. Στη συνέχεια εισάγουμε στον περιορισμό την μεταβλητή που αντιστοιχεί στον συγκεκριμένο πιλότο και συνεδρία. Έχοντας τελειώσει με αυτούς τους ιπτάμενους προσθέτουμε στον περιορισμό και τις μεταβλητές ανάθεσης που αντιστοιχούν στον εξεταζόμενο ιπτάμενο και τέλος προσθέτουμε και την μεταβλητή z εφόσον η μεταβλητή TSETείναι θετική.
- Επίσης εφόσον η μεταβλητή TSET είναι θετική προστίθεται ένας ακόμη περιορισμός. Σαρώνουμε ξανά τη λίστα με τους ιπταμένους και προσπαθούμε να εντοπίσουμε εκείνους που έχουν τα ίδια γνωρίσματα με

τους πιλότους που ανήκουν στο σύνολο το πλήθος του οποίου αντιπροσωπεύεται από τη μεταβλητή TSET. Έπειτα εισάγουμε στο αριστερό μέλος του περιορισμού τις μεταβλητές απόφασης που αντιπροσωπεύουν την ανάθεση αυτών των ιπταμένων στην εξεταζόμενη συνεδρία.

- Ελευθερώνουμε τις θέσεις μνήμης που είχαμε δεσμεύσει για τους περιορισμούς και τις μεταβλητές.

14. Η συνάρτηση Fix_Reverse_X2()

Και σε αυτή τη συνάρτηση, όπως και στις δύο προηγούμενες εκτελούμε ένα βρόγχο επανάληψης δύο φορές. Την πρώτη φορά για τους κυβερνήτες και τη δεύτερη για τους συγκυβερνήτες. Σε κάθε επανάληψη σαρώνεται η λίστα των εκπαιδευόμενων και αναζητούνται εκείνοι των οποίων η άδεια πτήσης λήγει τον μεθεπόμενο μήνα και των οποίων η θέση είναι όμοια με εκείνη για την οποία εκτελείται ο βρόγχος επανάληψης. Για κάθε έναν από αυτούς, σαρώνουμε τη λίστα με τις εφικτές του συνεδρίες και αναζητούμε εκείνες για τις οποίες ο εκάστοτε ιπτάμενος δεν έχει δηλώσει προτίμηση (priority = 0). Για καθεμία από αυτές τις συνεδρίες καλούμε τη συνάρτηση *find_NoOfSeniorIDs2*.

Η συνάρτηση *find_NoOfSeniorIDs2* παίρνει ως ορίσματα τον ιπτάμενο και τη συνεδρία που εξετάζεται και βρίσκει το πλήθος των αρχαιότερων ιπταμένων οι οποίοι έχουν ίδια θέση με τον υπό εξέταση ιπτάμενο, έχουν εφικτή την εξεταζόμενη συνεδρία με μηδενική τιμή προτεραιότητας και η πτητική τους ισχύς λήγει τον μεθεπόμενο μήνα.

Αφού η συνάρτηση *find_NoOfSeniorIDs2* βρει το πλήθος αυτών των ιπταμένων επιστρέφει την τιμή αυτή στη συνάρτηση *Fix_Reverse_X2()*. Σε περίπτωση που ο αριθμός αυτός είναι θετικός προστίθεται στο πρόβλημα ένας ακόμη περιορισμός ακολουθώντας την εξής διαδικασία:

Σαρώνουμε πάλι την PilotListκαι αναζητούμε ξανά τους ιπταμενους εκείνους το πλήθος των οποίων βρήκαμε με την συνάρτηση *find_NoOfSeniorIDs2*. Για κάθε έναν από αυτούς αναζητούμε στη λίστα των εφικτών συνεδριών του τη συνεδρία εκείνη που είναι ίδια με την εξεταζόμενη συνεδρία και που έχει μηδενική προτεραιότητα. Στη συνέχεια εισάγουμε στον περιορισμό την μεταβλητή που αντιστοιχεί σε αυτό τον πιλότο και αυτή τη συνεδρία. Έχοντας τελειώσει με αυτούς τους ιπτάμενους προσθέτουμε στον περιορισμό και τις μεταβλητές απόφασης που αντιστοιχούν στον εξεταζόμενο ιπτάμενο

15. Ησυνάρτηση Fix_Reverse_diffID_X2()

Η διαδικασία που ακολουθείται σε αυτή τη συνάρτηση είναι όμοια με εκείνη της συνάρτησης *Fix_Reverse_diffID_X1()*. Οι μόνες διαφορές είναι ότι τώρα:

- Αναζητούμε τους πιλότους που έχουν λήξη πτητικής ικανότητας τον μεθεπόμενο μήνα.
- Για κάθε ένα από αυτούς και για κάθε εφικτή του συνεδρία με τιμή προτεραιότητας 0 καλούμε τη *συνάρτηση NoOfdiffSeniorIDsX2*. Η συνάρτηση αυτή υπολογίζει το πλήθος των ιπταμένων που έχουν τα ακόλουθα χαρακτηριστικά: δεν ανήκουν στην ίδια θέση με τον εξεταζόμενο πιλότο, έχουν εφικτή την εξεταζόμενη συνεδρία με μηδενική τιμή προτεραιότητας και η πτητική τους ισχύς λήγει τον μεθεπόμενο μήνα και έχουν μεγαλύτερο βαθμό αρχαιότητας από τον εξεταζόμενο ιπτάμενο Η τιμή που υπολογίζει η συνάρτηση *NoOfdiffSeniorIDs2* επιστρέφεται στην συνάρτηση από την οποία και κλήθηκε.
- Εισάγουμε επίσης την βοηθητική μεταβλητή TSET η οποία σε αυτή τη συνάρτηση αντιπροσωπεύει το πλήθος των πιλότων που έχουν ίδια θέση με τον εξεταζόμενο ιπτάμενο και:
    - η πτητική τους ισχύ λήγει τον τρέχοντα μήνα και έχουν εφικτή την υπό εξέταση συνεδρία
    - η πτητική τους ισχύ λήγει τον επόμενο μήνα και έχουν εφικτή την εξεταζόμενη συνεδρία.
    - η πτητική τους ισχύ λήγει τον μεθεπόμενο μήνα και είτε έχουν εκφράσει προτίμηση για την υπό εξέταση συνεδρία είτε είναι νεότεροι από τον εξεταζόμενο ιπτάμενο.

16. Η συνάρτηση balance(Balancelevel)

Με τη συνάρτηση balance εισάγουμε στο πρόβλημα περιορισμούς με τους οποίους προσπαθούμε όσο είναι δυνατό να αναθέτουμε σε κάθε συνεδρία τον ίδιο αριθμό εκπαιδευόμενων γενικά αλλά και για κάθε θέση ειδικότερα. Έχοντας εκτελεστεί επιτυχώς η συνάρτηση Fix_Reverse_diffID_X2 () το πρόγραμμα επιστρέφει στην κύρια συνάρτηση main () όπου και ζητείται από το χρήστη της εφαρμογής να εισάγει έναν ακέραιο αριθμό ο οποίος αντιπροσωπεύει το βαθμό ισορροπίας που θέλουμε να έχει το σύστημά μας. Εφόσον η τιμή είναι 0 αυτό σημαίνει πως δεν εισάγεται κανένας περιορισμός ισορροπίας. Αν ο χρήστης δώσει την τιμή 1 δηλώνεται πως ο χρήστης προσπαθεί να διατηρήσει ισορροπία ως προς τον συνολικό αριθμό των ιπταμένων της κάθε συνεδρίας και τον αριθμό ιπταμένων του πιλοτηρίου και της καμπίνας (lightbalance). Αν πάλι η δοθείσα τιμή είναι 2 ο χρήστης προσπαθεί να

εξισορροπήσει όχι μόνο τον συνολικό αριθμό των ιπταμένων που ανατίθενται σε κάθε συνεδρία αλλά και τον αριθμό ιπταμένων ίδιας θέσης (κυβερνήτες, συγκυβερνήτες και προσωπικό καμπίνας) που συμμετέχουν σε κάθε συνεδρία(heavybalance). Εάν δοθεί μία τιμή διαφορετική από αυτές τις τρεις ζητείται από το χρήστη να εισάγει εκ νέου μια έγκυρη τιμή. Εφόσον η τιμή που δοθεί είναι έγκυρη καλείται η συνάρτηση balance η οποία έχει ως όρισμα την τιμή που δόθηκε από το χρήστη και η οποία ορίζεται με τη μεταβλητή Balancelevel. Η συνάρτηση ελέγχει την τιμή της μεταβλητής balancelevel.

Αν η τιμή είναι 2 τότε εκτελείται ένας βρόγχος επανάληψης τέσσερις φορές. Μία για όλους τους ιπταμένους ανεξαρτήτως θέσης, μία για τους κυβερνήτες, μια για τους συγκυβερνήτες και μία για το πλήρωμα καμπίνας. Σε κάθε επανάληψη σαρώνεται η λίστα με τις εκπαιδευτικές συνεδρίες του μήνα και για κάθε μια από αυτές σαρώνουμε την λίστα των ιπταμένων και βρίσκουμε εκείνους που έχουν εφικτή τη συγκεκριμένη συνεδρία και θέση ίδια με αυτή που ελέγχει ο βρόγχος επανάληψης. Αν δεν βρούμε κάποιον με αυτές τις ιδιότητες προχωράμε στην επόμενη συνεδρία της SessionList. Αν όμως συμβεί το αντίθετο τότε προσθέτουμε στο αριστερό μέλος του περιορισμού τις μεταβλητές απόφασης που αντιπροσωπεύουν την ανάθεση των ιπταμένων στη συγκεκριμένη συνεδρία.

Έπειτα εκτελούμε ένα βρόγχο επανάληψης για όλες εκείνες τις συνεδρίες της SessionList που έχουν αύξων αριθμό μεγαλύτερο από την εξεταζόμενη συνεδρία. Σε κάθε επανάληψη αναζητούμε εκείνους τους πιλότους που έχουν εφικτή τη συγκεκριμένη συνεδρία και έχουν θέση ίδια με αυτή που ορίζεται από τον εξωτερικό βρόγχο επανάληψης και εισάγουμε τις αντίστοιχες μεταβλητές απόφασης στον περιορισμό. Σε περίπτωση που δεν βρούμε κάποιον ιπτάμενο τότε ο εσωτερικός βρόγχος επανάληψης συνεχίζει με την επόμενη συνεδρία. Έπειτα εισάγονται στον περιορισμό δύο μεταβλητές απόφασης μία για κάθε συνεδρία.

Σε περίπτωση που το balancelevel που εισάγεται από το χρήστη είναι 1 τότε η διαδικασία που ακολουθείται είναι ακριβώς η ίδια με αυτή που περιγράψαμε πιο πάνω με τη διαφορά ότι ο εξωτερικός βρόγχος επανάληψης εκτελείται τρεις φορές.Μία για όλους τους ιπταμένους ανεξαρτήτως θέσης, μία για το πλήρωμα του πιλοτηρίου και μία για το πλήρωμα καμπίνας.

17. Η συνάρτηση DoPenalty(PList)

Έχοντας εκτελεστεί επιτυχώς η συνάρτηση balance το πρόγραμμα επιστρέφει στην κύρια συνάρτηση main() και σαρώνει την PenaltyList προκειμένου να βρει το τελευταίο της στοιχείο που αντιστοιχεί στη μεταβλητή w. Φτάνοντας στο τέλος της

λίστας καλείται η συνάρτηση DoPenalty η οποία αλλάζει τον συντελεστή ορισμένων μεταβλητών απόφασης στην αντικειμενική συνάρτηση, επιλύει το πρόβλημα και δίνει τις τιμές αυτών των μεταβλητών που οδηγούν στη βέλτιστη λύση. Πιο συγκεκριμένα:

- Ορίζουμε αρχικά την παράμετρο numpenalties που δηλώνει κατά προσέγγιση τον αριθμό των μεταβλητών απόφασης για τις οποίες θα αλλάξουμε συντελεστή και θα επιλύσουμε το πρόβλημα. Ξεκινώντας από τον τελευταίο κόμβο της λίστας ,για τον οποίον ορίζουμε ένα δείκτη (index) με τιμή ίση με τη μονάδα,προχωράμε προς τα αριστερά της λίστας αυξάνοντας το δείκτη κατά μία μονάδα για κάθε κόμβο της λίστας που σαρώνουμε. Σταματάμε όταν ο δείκτης γίνει ίσος με την παράμετρο numpenalties.Στη συνέχεια ελέγχουμε αν ο κόμβος που βρίσκεται αριστερά αυτού στον οποίο σταματήσαμε αναφέρεται στον ίδιο ιπτάμενο. Αν συμβαίνει κάτι τέτοιο μετακινούμαστε προς τα αριστερά όσους κόμβους χρειάζεται προκειμένου να συμπεριληφθούν όλοι οι κόμβοι που ανήκουν στον ίδιο ιπτάμενο.
- Έπειτα πραγματοποιούμε από τον τελευταίο κόμβο που καταμετρήσαμε μέχρι και τον αρχικό την αλλαγή των συντελεστών των μεταβλητών που αντιστοιχούν στους κόμβους που καταμετρήσαμε παραπάνω με τη λογική που παρουσιάσαμε σε προηγούμενη ενότητα. Η αλλαγή των συντελεστών γίνεται χρησιμοποιώντας τη συνάρτηση CPXchgcoef η οποία έχει ως ορίσματα τον δείκτη για το περιβάλλον της cplex, το δείκτη για το πρόβλημα, έναν ακέραιο που δηλώνει τον δείκτη της γραμμής στην οποία βρίσκεται ο συντελεστής, τον δείκτη της μεταβλητής της οποίας τον συντελεστή θα αλλάξουμε και την νέα τιμή του συντελεστή.
- Μετά την αλλαγή των συντελεστών επιλύουμε το πρόβλημα και στη συνέχεια σαρώνουμε πάλι τους ίδιους κόμβους. Όταν εντοπίζουμε έναν κόμβο με RecType ‘P’ ο οποίος δεν αντιστοιχεί στη χαμηλότερη επιθυμία του ιπταμένου και η αντίστοιχη μεταβλητή απόφασης είναι μονάδα εισάγουμε έναν νέο περιορισμό ισότητας με αριστερό μέλος την μεταβλητή αυτή και δεξιό μέλος 1. Έπειτα κλειδώνουμε εισάγοντας περιορισμό και τη μεταβλητή απόφασης w που αντιστοιχεί σε κόμβο με RecType ‘M’. Στη συνέχεια αλλάζουμε πάλι τους συντελεστές αυτών των κόμβων πάλι με 0.
- Μετά την επιτυχημένη ολοκλήρωση της η DoPenalty επιστρέφει ένα δείκτη ο οποίος δείχνει σε εκείνο τον κόμβο που βρίσκεται αριστερά του τελευταίου κόμβου του οποίου ο συντελεστής άλλαξε στην τελευταία εκτέλεση της συνάρτησης. Η συνάρτηση DoPenalty καλείται όσες φορές χρειάζεται προκειμένου να πραγματοποιηθεί η επίλυση για όλους τους κόμβους.

Αξίζει να σημειώσουμε ότι μεταφερόμαστε στον τελευταίο κόμβο της PenaltyList πριν καλέσουμε την DoPenalty έτσι ώστε ο κόμβος αυτό να συμπεριλαμβάνεται στην πρώτη επανάληψη και να δοθεί εξαρχής μια τιμή στην μεταβλητή w γιατί αυτό που επιδιώκεται πάνω από όλα είναι η μείωση των ιπτάμενων που μένουν εκτός εκπαίδευσης.

18. Η συνάρτηση solve_evaluate_results

Με αυτή την υπορουτίνα επιλύουμε το πρόβλημα και αξιολογούμε τα αποτελέσματα που προκύπτουν. Οι εντολές που εκτελούνται είναι:

- CPXmipopt η οποία χρησιμοποιείται για να βρούμε μια λύση στο πρόβλημα του Μεικτού Ακέραιου Προγραμματισμού
- CPXgetstat η οποία μας δίνει πρόσβαση στη λύση του προβλήματος εάν υπάρχει εφικτή λύση.
- CPXgetobjval η οποία υπολογίζει την τιμή της αντικειμενικής συνάρτησης.

Έπειτα από την εκτέλεση αυτών των εντολών εκτυπώνουμε σε ένα αρχείο τα αποτελέσματα που παρουσιάζουν τις συνεδρίες στις οποίες ανατίθενται οι ιπτάμενοι, ποιοι ιπτάμενοι δεν ανατίθενται για εκπαίδευση (αν υπάρχουν) αλλά και ποια γλώσσα ανατίθεται σε κάθε εκπαιδευτική συνεδρία.

19. clock()

Με την εντολή αυτή ορίζουμε το σημείο παύσης της καταγραφής και καταμέτρησης του χρόνου επίλυσης του προβλήματος.

20. Ησυνάρτηση free_problem

Με την εκτέλεση της εντολής CPXfreeprob αφαιρούμε το πρόβλημα της cplex από το περιβάλλον της cplex και ελευθερώνουμε τη μνήμη που χρειαστήκαμε για την δημιουργία και την εκτέλεση του κώδικα. Με την εντολή CPXcloseCPLEX ελευθερώνονται όλες οι δομές δεδομένων που σχετίζονται με τη CPLEX.

6. Εφαρμογή του κώδικα

Στη συνέχεια παρουσιάζουμε δύο προβλήματα βελτιστοποίησης για την επίλυση των οποίων χρησιμοποιήσαμε τον κώδικα που περιγράψαμε στο κεφάλαιο 4. Με τα παραδείγματα αυτά διαπιστώνουμε αν ο κώδικας αναπαράγει σωστά το εκάστοτε πρόβλημα. Επίσης, μετά την επίλυση αυτών των προβλημάτων συγκρίνουμε τους χρόνουςεπίλυσης του κάθε προβλήματος για διαφορετικά επίπεδα ισορροπίας.

6.1. Πρόβλημα 1

Το πρώτο πρόβλημα αφορά την ανάθεση δεκαπέντε (15) ιπτάμενων σε τέσσερις (4) εκπαιδευτικές συνεδρίες. Στον πίνακα παρουσιάζονται τα χαρακτηριστικά κάθε πιλότου αλλά και τα χαρακτηριστικά κάθε συνεδρίας.

	Συνολική χωρητικότητα	***Χωρητικότητα ιπτάμενων πιλοτηρίου***	***Χωρητικότητα προσωπικού καμπίνας***	***Χωρητικότητα πιλότων***	***Χωρητικότητα συγκυβερνητών***	***Γλώσσα***
Για κάθε συνεδρία	*20*	*14*	*10*	*8*	*8*	*Δεν έχει καθοριστεί*

Πίνακας 6.1 : Χαρακτηριστικά συνεδριών του προβλήματος 1

N	POSITION	SENIORITY	EXPIRATION	LANGUAGE
1	1 (CP)	12	0	0(both)
2	3(CABIN)	8	0	0(both)
3	2(FO)	7	1	0(both)
4	2(FO)	4	2	1(FR)
5	1(CP)	2	1	2(EN)
6	3(CABIN)	1	0	0(both)
7	2(FO)	3	0	1(FR)
8	3(CABIN)	5	0	2(EN)
9	3(CABIN)	6	0	0(both)
10	2(FO)	9	2	0(both)
11	3(CABIN)	10	0	0(both)
12	3(CABIN)	11	0	0(both)
13	3(CABIN)	13	0	1(FR)
14	1(CP)	14	1	1(FR)
15	1(CP)	15	1	2(EN)

Πίνακας 6.2: Χαρακτηριστικά ιπτάμενων του προβλήματος 1

S / N	1	2	3	4	Πλήθος εφικτών συνεδριών
1		0			1
2	0				1
3		0		0	2
4		1		2	2
5		1			1
6	0	1			2
7	0		0		2
8	0		0		2
9	1			0	2
10			0	0	2
11		1			1
12	1				1
13		0	1		2
14	0			1	2
15	1		0		2

Πίνακας 6.3 :Λίστα εφικτών συνεδριών και προτιμήσεων των ιπτάμενων του προβλήματος 1

Στη συνέχεια παρουσιάζεται η αντιστοίχιση των μεταβλητών ανάθεσης των ιπταμένων του μαθηματικού μοντέλου με αυτές που παράγει ο κώδικας κατά την εκτέλεση του.

Μεταβλητή απόφασης μαθηματικού μοντέλου	Μεταβλητή απόφασης του προγράμματος	Μεταβλητή απόφασης μαθηματικού μοντέλου	Μεταβλητή απόφασης του προγράμματος
x_{12}	x_1	$x_{10,4}$	x_{17}
x_{21}	x_2	$x_{11,2}$	x_{18}
x_{32}	x_3	$x_{12.1}$	x_{19}

x_{34}	x_4	$x_{13,2}$	x_{20}
x_{42}	x_5	$x_{13,3}$	x_{21}
x_{44}	x_6	$x_{14,1}$	x_{22}
x_{52}	x_7	$x_{14,4}$	x_{23}
x_{61}	x_8	$x_{15,1}$	x_{24}
x_{62}	x_9	$x_{15,3}$	x_{25}
x_{71}	x_{10}	y_3	x_{26}
x_{73}	x_{11}	y_4	x_{27}
x_{81}	x_{12}	y_5	x_{28}
x_{83}	x_{13}	y_{10}	x_{29}
x_{91}	x_{14}	y_{14}	x_{30}
x_{94}	x_{15}	y_{15}	x_{31}
$x_{10,3}$	x_{16}	w	x_{32}

Πίνακας 6.4 : Αντιστοίχιση των μεταβλητών ανάθεσης του μαθηματικού μοντέλου με αυτές του κώδικα για το πρόβλημα 1

Για balancelevel = 0

Η αντικειμενική συνάρτηση είναι :

minimize0 x_{24} + 1 x_{31}+ 2 x_{25}+ 0 x_{23}+ 3 x_{30}+ 6 x_{22}+ 0 x_{21}+ 9 x_{20}+ 0 x_1+ 0 x_{19}+ 0 x_{18}+ 0 x_{29} +18 x_{16}+ 18 x_{17} +0 x_2+ 0 x_{26}+ 36 x_3+36 x_4+ 0 x_{14}+ 72 x_{15}+ 0 x_{12}+ 0 x_{13}+ 0 x_5+ 144 x_6+ 288 x_{27} 0 x_{10}+ 0 x_{11} + 0 x_7+ 432 x_{28}+ 0x_9 +864 x_8 +1728 x_{32}

Οι μεταβλητές απόφασης τοποθετήθηκαν στην αντικειμενική συνάρτηση ακριβώς όπως ταξινομήθηκαν στην PenaltyList, ξεκινώντας δηλαδή από το νεότερο ιπτάμενο και τοποθετώντας τις μεταβλητές απόφασης του κάθε ιπταμένου με το μεγαλύτερο prior προς τα αριστερά της λίστας.

Οι περιορισμοί του προβλήματος είναι οι εξής:

- ✓ *Περιορισμοί ανάθεσης*

MC1: x1 = 1

MC2: x2 = 1

MC3: x3 + x4 + x26 = 1

MC4: x5 + x6 + x27 = 1

MC5: x7 + x28 = 1

MC6: x8 + x9 = 1

MC7: x10 + x11 = 1

MC8: x12 + x13 = 1

MC9: x14 + x15 = 1

MC10: x16 + x17 + x29 = 1

MC11: x18 = 1

MC12: x19 = 1

MC13: x20 + x21 = 1

MC14: x22 + x23 + x30 = 1

MC15: x24 + x25 + x31 = 1

- ✓ *Περιορισμοί χωρητικότητας*

TOTALCAP_session_1: x2 + x8 + x10 + x12 + x14 + x19 + x22 + x24 <= 20

TOTALCAP_session_2: x1 + x3 + x5 + x7 + x9 + x18 + x20 <= 20

TOTALCAP_session_3: x11 + x13 + x16 + x21 + x25 <= 20

TOTALCAP_session_4: x4 + x6 + x15 + x17 + x23 <= 20

COCKPIT_ses_1: x10 + x22 + x24 <= 14

COCKPIT_ses_2: x1 + x3 + x5 + x7 <= 14

COCKPIT_ses_3: x11 + x16 + x25 <= 14

COCKPIT_ses_4: x4 + x6 + x17 + x23 <= 14

Cabin_ses_1: x2 + x8 + x12 + x14 + x19 <= 10

Cabin_ses_2: x9 + x18 + x20 <= 10

Cabin_ses_3: x13 + x21 <= 10

Cabin_ses_4: x15 <= 10

CP'S_ses_1: x22 + x24 <= 8

CP'S_ses_2: x1 + x7 <= 8

CP'S_ses_3: x25 <= 8

CP'S_ses_4: x23 <= 8

FO's_ses_1: x10 <= 8

FO's_ses_2: x3 + x5 <= 8

FO's_ses_3: x11 + x16 <= 8

FO's_ses_4: x4 + x6 + x17 <= 8

- ✓ *Περιορισμός ιπταμένων που παραμένουν εκτός εκπαίδευσης*

NOEMS: - x26 - x27 - x28 - x29 - x30 - x31 + x32 = 0

- ✓ *Περιορισμοί ανάθεσης γλώσσας*

LANG1ses1: x10 + x22 - 2 x33 <= 0

LANG2ses1: x12 + x24 + 2 x33 <= 2

LANG1ses2: x5 + x20 - 2 x34 <= 0

LANG2ses2: x7 + x34 <= 1

LANG1ses3: x11 + x21 - 2 x35 <= 0

LANG2ses3: x13 + x25 + 2 x35 <= 2

LANG1ses4: x6 + x23 - 2 x36 <= 0

- ✓ *Περιορισμοί αντίστροφης αρχαιότητας*

FixRevID3SES4: x3 + x4 - x17 >= 0

FixRevdifX1ID15SES3: - x16 + x24 + x25 >= 0

Ο χρόνος επίλυσης του προβλήματος είναι 1,313sec και η τιμή της αντικειμενικής συνάρτησης είναι 198. Τα αποτελέσματα που προέκυψαν φαίνονται στους πίνακες

Ιπτάμενος	**Συνεδρία στην οποία ανατίθεται**
1	2
2	1
3	4
4	4
5	2
6	2
7	3
8	1
9	1
10	4
11	2
12	1
13	3
14	4
15	1

Πίνακας 6.5: Αποτελέσματα ανάθεσης ιπτάμενων σε συνεδρίες για το πρόβλημα 1

Εκπαιδευτική συνεδρία	Γλώσσα ανάθεσης
1	0(EN)
2	0(EN)
3	1(FR)
4	1(FR)

Πίνακας 6.6: Αποτελέσματα ανάθεσης γλώσσας στις εκπαιδευτικές συνεδρίες του προβλήματος 1

Για balancelevel=1

Η αντικειμενική συνάρτηση παραμένει ίδια μόνο που τώρα προστίθενται και οι μεταβλητές απόφασης με δείκτες από 37 έως 72 και συντελεστή αντικειμενικής συνάρτησης 0,05.

Επίσης προστίθενται στο μαθηματικό μοντέλο και οι παρακάτω περιορισμοί ισορροπίας:

balance_constraintPOS0SES1SES2: - x1 + x2 - x3 - x5 - x7 + x8 - x9 + x10 + x12 + x14 - x18 + x19 - x20 + x22 + x24 + x37- x38 = 0

balance_constraintPOS0SES1SES3: x2 + x8 + x10 - x11 + x12 - x13 + x14 - x16 + x19 - x21 + x22 + x24 - x25 + x39 - x40 = 0

balance_constraintPOS0SES1SES4: x2 - x4 - x6 + x8 + x10 + x12 + x14 - x15 - x17 + x19 + x22 - x23 + x24 + x41 - x42 = 0

balance_constraintPOS0SES2SES3: x1 + x3 + x5 + x7 + x9 - x11 - x13 - x16 + x18 + x20 - x21 - x25 + x43 - x44 = 0

balance_constraintPOS0SES2SES4: x1 + x3 - x4 + x5 - x6 + x7 + x9 - x15 - x17 + x18 + x20 - x23 + x45 - x46 = 0

balance_constraintPOS0SES3SES4: - x4 - x6 + x11 + x13 - x15 + x16 - x17 + x21 - x23 + x25 + x47 - x48 = 0

balance_constraintPOS1SES1SES2: - x1 - x3 - x5 - x7 + x10 + x22 + x24 + x49 - x50 = 0

balance_constraintPOS1SES1SES3: x10 - x11 - x16 + x22 + x24 - x25 + x51 - x52 = 0

balance_constraintPOS1SES1SES4: - x4 - x6 + x10 - x17 + x22 - x23 + x24 + x53 - x54 = 0

balance_constraintPOS1SES2SES3: x1 + x3 + x5 + x7 - x11 - x16 - x25 + x55

- x56 = 0

balance_constraintPOS1SES2SES4: x1 + x3 - x4 + x5 - x6 + x7 - x17 - x23 + x57 - x58 = 0

balance_constraintPOS1SES3SES4: - x4 - x6 + x11 + x16 - x17 - x23 + x25 + x59 - x60 = 0

balance_constraintPOS2SES1SES2: x2 + x8 - x9 + x12 + x14 - x18 + x19 - x20 + x61 - x62 = 0

balance_constraintPOS2SES1SES3: x2 + x8 + x12 - x13 + x14 + x19 - x21 + x63 - x64 = 0

balance_constraintPOS2SES1SES4: x2 + x8 + x12 + x14 - x15 + x19 + x65 - x66 = 0

balance_constraintPOS2SES2SES3: x9 - x13 + x18 + x20 - x21 + x67 - x68 = 0

balance_constraintPOS2SES2SES4: x9 - x15 + x18 + x20 + x69 - x70 = 0

balance_constraintPOS2SES3SES4: x13 - x15 + x21 + x71 - x72 = 0

Από την επίλυση του προβλήματος προκύπτουν τα ίδια αποτελέσματα με εξαίρεση το γεγονός πως ο δέκατος ιπτάμενος δεν ανατίθεται πλέον στην τέταρτη συνεδρία αλλά στην τρίτη χωρίς να παραβιάζεται κάποιος περιορισμός και χωρίς να δυσαρεστείται ο ιπτάμενος(αφού για καμία από τις 2 συνεδρίες δεν είχε εκφράσει επιθυμία). Η τιμή της αντικειμενικής συνάρτησης είναι 199,3 ενώ ο χρόνος επίλυσης του προβλήματος αυξήθηκε στα 1,875sec γεγονός που συμβαίνει εξαιτίας της αύξησης του αριθμού των μεταβλητών απόφασης από 36 σε 72.

Για balancelevel = 2

Σε σχέση με την περίπτωση όπου balancelevel =0, η αντικειμενική συνάρτηση παραμένει ίδια μόνο που τώρα προστίθενται και οι μεταβλητές απόφασης με δείκτες από 37 έως 84 και συντελεστή αντικειμενικής συνάρτησης 0,05.

Επίσης προστίθενται στο μαθηματικό μοντέλο και οι παρακάτω περιορισμοί ισορροπίας

balance_constraintPOS0SES1SES2: - x1 + x2 - x3 - x5 - x7 + x8 - x9 + x10 + x12 + x14 - x18 + x19 - x20 + x22 + x24 + x37 - x38 = 0

balance_constraintPOS0SES1SES3: x2 + x8 + x10 - x11 + x12 - x13 + x14 - x16 + x19 - x21 + x22 + x24 - x25 + x39 - x40 = 0

balance_constraintPOS0SES1SES4: x2 - x4 - x6 + x8 + x10 + x12 + x14 - x15

- x17 + x19 + x22 - x23 + x24 + x41 - x42 = 0

balance_constraintPOS0SES2SES3: x1 + x3 + x5 + x7 + x9 - x11 - x13 - x16 + x18 + x20 - x21 - x25 + x43 - x44 = 0

balance_constraintPOS0SES2SES4: x1 + x3 - x4 + x5 - x6 + x7 + x9 - x15 - x17 + x18 + x20 - x23 + x45 - x46 = 0

balance_constraintPOS0SES3SES4: - x4 - x6 + x11 + x13 - x15 + x16 - x17 + x21 - x23 + x25 + x47 - x48 = 0

balance_constraintPOS1SES1SES2: - x1 - x7 + x22 + x24 + x49 - x50 = 0

balance_constraintPOS1SES1SES3: x22 + x24 - x25 + x51 - x52 = 0

balance_constraintPOS1SES1SES4: x22 - x23 + x24 + x53 - x54 = 0

balance_constraintPOS1SES2SES3: x1 + x7 - x25 + x55 - x56 = 0

balance_constraintPOS1SES2SES4: x1 + x7 - x23 + x57 - x58 = 0

balance_constraintPOS1SES3SES4: - x23 + x25 + x59 - x60 = 0

balance_constraintPOS2SES1SES2: - x3 - x5 + x10 + x61 - x62 = 0

balance_constraintPOS2SES1SES3: x10 - x11 - x16 + x63 - x64 = 0

balance_constraintPOS2SES1SES4: - x4 - x6 + x10 - x17 + x65 - x66 = 0

balance_constraintPOS2SES2SES3: x3 + x5 - x11 - x16 + x67 - x68 = 0

balance_constraintPOS2SES2SES4: x3 - x4 + x5 - x6 - x17 + x69 - x70 = 0

balance_constraintPOS2SES3SES4: - x4 - x6 + x11 + x16 - x17 + x71 - x72 = 0

balance_constraintPOS3SES1SES2: x2 + x8 - x9 + x12 + x14 - x18 + x19 - x20 + x73 - x74 = 0

balance_constraintPOS3SES1SES3: x2 + x8 + x12 - x13 + x14 + x19 - x21 + x75 - x76 = 0

balance_constraintPOS3SES1SES4: x2 + x8 + x12 + x14 - x15 + x19 + x77 - x78 = 0

balance_constraintPOS3SES2SES3: x9 - x13 + x18 + x20 - x21 + x79 - x80 = 0

balance_constraintPOS3SES2SES4: x9 - x15 + x18 + x20 + x81 - x82 = 0

balance_constraintPOS3SES3SES4: x13 - x15 + x21 + x83 - x84 = 0

Συγκρίνοντας τα αποτελέσματα για balancelevel =2 με αυτά που προέκυψαν για balancelevel = 1 παρατηρούμε πως τα αποτελέσματα που αφορούν την ανάθεση των ιπτάμενων είναι τα ίδια . Η τιμή της αντικειμενικής συνάρτησης αυξήθηκε στις 199,7 μονάδες ενώ ο χρόνος επίλυσης του προβλήματος αυξήθηκε στα 1,953sec γεγονός που συμβαίνει εξαιτίας της αύξησης του αριθμού των μεταβλητών απόφασης από 72 σε 84..

Τα αποτελέσματα από την εκτέλεση του ίδιου προβλήματος για διαφορετικά επίπεδα ισορροπίας συνοψίζονται στον παρακάτω πίνακα.

balance level	**0**	**1**	**2**
Χρόνος επίλυσης	1.313sec	1.875sec	1.953sec
Σύνολο μεταβλητών απόφασης	36	72	84
Τιμή αντικειμενικής συνάρτησης	198	199.3	199.7

Πίνακας 6.7 : Σύγκριση αποτελεσμάτων από την επίλυση του προβλήματος 1 για διαφορετικά επίπεδα ισορροπίας

6.2. Πρόβλημα 2

Το δεύτερο πρόβλημα αφορά την ανάθεση δεκαπέντε (20) ιπτάμενων σε τέσσερις (6) εκπαιδευτικές συνεδρίες. Στον πίνακα παρουσιάζονται τα χαρακτηριστικά κάθε πιλότου αλλά και τα χαρακτηριστικά κάθε συνεδρίας.

	Συνολική χωρητικότητα	**Χωρητικότητα ιπτάμενων πιλοτηρίου**	**Χωρητικότητα προσωπικού καμπίνας**	**Χωρητικότητα πιλότων**	**Χωρητικότητα συγκυβερνητών**	**Γλώσσα**
Για κάθε συνεδρία	20	14	10	8	8	Δεν έχει καθοριστεί

Πίνακας 6.8 : Χαρακτηριστικά συνεδριών του προβλήματος 2

N	POSITION	SENIORITY	EXPIRATION	LANGUAGE
1	1(CP)	20	2	1(FR)
2	2(FO)	19	1	2(EN)
3	3(CABIN)	18	0	0(both)
4	1(CP)	17	2	0(both)
5	3(CABIN)	16	0	2(EN)
6	1(CP)	12	0	2(EN)
7	3(CABIN)	8	0	1(FR)
8	3(CABIN)	7	0	2(EN)
9	1(CP)	4	2	2(EN)

10	2(FO)	2	0	0(both)
11	3(CABIN)	1	0	0(both)
12	2(FO)	3	2	2(EN)
13	2(FO)	5	1	0(both)
14	3(CABIN)	6	0	0(both)
15	2(FO)	9	0	0(both)
16	1(CP)	10	2	2(EN)
17	3(CABIN)	11	2	1(FR)
18	1(CP)	13	0	0(both)
19	2(FO)	14	0	1(FR)
20	1(CP)	15	2	2(EN)

Πίνακας 6.9: Χαρακτηριστικά ιπτάμενων του προβλήματος 2

S \ N	1	2	3	4	5	6	Πλήθος εφικτών συνεδριών
1	0	0					2
2	1			2			2
3						0	1
4						0	1
5	1	0					2
6				0		0	2
7				0	0		2
8		0		0		0	3
9	0	1		2			3
10			1				1
11					0		1
12				1			1
13			0	1			2
14						1	1
15	1			2			2

16				1			1
17	0	0			0		3
18				0	1		2
19			1			0	2
20		1	0				2

Πίνακας 6.10 :Λίστα εφικτών συνεδριών και προτιμήσεων των ιπτάμενων του προβλήματος 2

Στη συνέχεια παρουσιάζεται η αντιστοίχιση των μεταβλητών του μαθηματικού μοντέλου με αυτές που παράγει ο κώδικας κατά την εκτέλεση του.

Μεταβλητή απόφασης μαθηματικού μοντέλου	Μεταβλητή απόφασης του προγράμματος	Μεταβλητή απόφασης μαθηματικού μοντέλου	Μεταβλητή απόφασης του προγράμματος
x_{11}	x_1	$x_{14,6}$	x_{24}
x_{12}	x_2	$x_{15,1}$	x_{25}
x_{21}	x_3	$x_{15,4}$	x_{26}
x_{24}	x_4	$x_{16,4}$	x_{27}
x_{36}	x_5	$x_{17,1}$	x_{28}
x_{46}	x_6	$x_{17,2}$	x_{29}
x_{51}	x_7	$x_{17,5}$	x_{30}
x_{52}	x_8	$x_{18,4}$	x_{31}
x_{64}	x_9	$x_{18,5}$	x_{32}
x_{66}	x_{10}	$x_{19,3}$	x_{33}
x_{74}	x_{11}	$x_{19,6}$	x_{34}
x_{75}	x_{12}	$x_{20,2}$	x_{35}
x_{82}	x_{13}	$x_{20,3}$	x_{36}
x_{84}	x_{14}	y_1	x_{37}
x_{86}	x_{15}	y_2	x_{38}
x_{91}	x_{16}	y_4	x_{39}
x_{92}	x_{17}	y_9	x_{40}
x_{94}	x_{18}	y_{12}	x_{41}
$x_{10,3}$	x_{19}	y_{13}	x_{42}
$x_{11,5}$	x_{20}	y_{16}	x_{43}
$x_{12,4}$	x_{21}	y_{20}	x_{44}

$x_{13,3}$	x_{22}	w	x_{45}
$x_{13,4}$	x_{23}		

Πίνακας 6.11 : Αντιστοίχιση των μεταβλητών ανάθεσης του μαθηματικού μοντέλου με αυτές του κώδικα για το πρόβλημα 2

Για balancelevel = 0

Η αντικειμενική συνάρτηση είναι :

Minimize $0\,x_{36} + x_1 + x_2 + 0\,x_3 + 2\,x_4 + 4\,x_{38} + 0x_5 + 0\,x_{39} + 6\,x_6 + 0\,x_7 + 12\,x_8 + 0\,x_{35} + 24\,x_{44} + 48\,x_{36} + 0\,x_{33} + 72\,x_{34} + 0\,x_{32} + 144\,x_{31} + 0\,x_{10} + 0\,x_9 + 0\,x_{30} + 0x_{29} + 0x_{28} + 0\,x_{27} + 288\,x_{43} + 0\,x_{25} + 576\,x_{26} + 0x_{12} + 0x_{11} + 0\,x_{15} + 0\,x_{14} + 0x_{13} + 0\,x_{24} + 0x_{23} + 1152\,x_{42} + 2304\,x_{22} + 0x_{17} + 3456\,x_{18} + 6912\,x_{40} + 10368\,x_{16} + 0x_{21} + 13824\,x_{41} + 0x_{19} + 0x_{20} + 27648\,x_{45}$

Οι μεταβλητές απόφασης τοποθετήθηκαν στην αντικειμενική συνάρτηση ακριβώς όπως ταξινομήθηκαν στην PenaltyList, ξεκινώντας δηλαδή από το νεότερο ιπτάμενο και τοποθετώντας τις μεταβλητές απόφασης του κάθε ιπταμένου με το μεγαλύτερο prior προς τα αριστερά της λίστας.

Οι περιορισμοί του προβλήματος είναι οι εξής:

- ✓ *Περιορισμοί ανάθεσης*

MC1: x1 + x2 + x37 = 1
MC2: x3 + x4 + x38 = 1
MC3: x5 = 1
MC4: x6 + x39 = 1
MC5: x7 + x8 = 1
MC6: x9 + x10 = 1
MC7: x11 + x12 = 1
MC8: x13 + x14 + x15 = 1
MC9: x16 + x17 + x18 + x40 = 1
MC10: x19 = 1
MC11: x20 = 1
MC12: x21 + x41 = 1
MC13: x22 + x23 + x42 = 1
MC14: x24 = 1
MC15: x25 + x26 = 1
MC16: x27 + x43 = 1

MC17: x28 + x29 + x30 = 1

MC18: x31 + x32 = 1

MC19: x33 + x34 = 1

MC20: x35 + x36 + x44 = 1

- ✓ *Περιορισμοί χωρητικότητας*

TOTALCAP_session_1: x1 + x3 + x7 + x16 + x25 + x28 <= 20

TOTALCAP_session_2: x2 + x8 + x13 + x17 + x29 + x35 <= 20

TOTALCAP_session_3: x19 + x22 + x33 + x36 <= 20

TOTALCAP_session_4: x4 + x9 + x11 + x14 + x18 + x21 + x23 + x26 + x27 + x31 <= 20

TOTALCAP_session_5: x12 + x20 + x30 + x32 <= 20

TOTALCAP_session_6: x5 + x6 + x10 + x15 + x24 + x34 <= 20

COCKPIT_ses_1: x1 + x3 + x16 + x25 <= 14

COCKPIT_ses_2: x2 + x17 + x35 <= 14

COCKPIT_ses_3: x19 + x22 + x33 + x36 <= 14

COCKPIT_ses_4: x4 + x9 + x18 + x21 + x23 + x26 + x27 + x31 <= 14

COCKPIT_ses_5: x32 <= 14

COCKPIT_ses_6: x6 + x10 + x34 <= 14

Cabin_ses_1: x7 + x28 <= 10

Cabin_ses_2: x8 + x13 + x29 <= 10

Cabin_ses_4: x11 + x14 <= 10

Cabin_ses_5: x12 + x20 + x30 <= 10

Cabin_ses_6: x5 + x15 + x24 <= 10

CP'S_ses_1: x1 + x16 <= 8

CP'S_ses_2: x2 + x17 + x35 <= 8

CP'S_ses_3: x36 <= 8

CP'S_ses_4: x9 + x18 + x27 + x31 <= 8

CP'S_ses_5: x32 <= 8

CP'S_ses_6: x6 + x10 <= 8

FO's_ses_1: x3 + x25 <= 8

FO's_ses_3: x19 + x22 + x33 <= 8

FO's_ses_4: x4 + x21 + x23 + x26 <= 8

FO's_ses_6: x34 <= 8

- ✓ *Περιορισμός ιπταμένων που παραμένουν εκτός εκπαίδευσης*

NOEMS: - x37 - x38 - x39 - x40 - x41 - x42 - x43 - x44 + x45 = 0

- ✓ *Περιορισμοί ανάθεσης γλώσσας*

LANG1ses1: x1 + x28 - 2 x46 <= 0

LANG2ses1: x3 + x7 + x16 + 3 x46 <= 3

LANG1ses2: x2 + x29 - 2 x47 <= 0

LANG2ses2: x8 + x13 + x17 + x35 + 4 x47 <= 4

LANG1ses3: x33 - x48 <= 0

LANG2ses3: x36 + x48 <= 1

LANG1ses4: x11 - x49 <= 0

LANG2ses4: x4 + x9 + x14 + x18 + x21 + x27 + 6 x49 <= 6

LANG1ses5: x12 + x30 - 2 x50 <= 0

LANG1ses6: x34 - x51 <= 0

LANG2ses6: x10 + x15 + 2 x51 <= 2

- ✓ *Περιορισμοί αντίστροφης αρχαιότητας*

FixRevdifX1ID13SES3: x22 + x23 - x36 + x52 >= 0

FixRevIzvarID13SES3: x19 + x33 - 8 x52 >= 0

FixRevIDII1SES1: x1 + x2 - x16 >= 0

Στους περιορισμούς δεν συμπεριλαμβάνονται οι περιορισμοί ισορροπίας γιατί όταν ζητήθηκε δηλώσαμε μηδενικό balancelevel. Ο χρόνος επίλυσης του προβλήματος είναι 1,031sec και η τιμή της αντικειμενικής συνάρτησης είναι 21. Τα αποτελέσματα που προέκυψαν φαίνονται στους πίνακες

Ιπτάμενος	Συνεδρία στην οποία ανατίθεται	Ιπτάμενος	Συνεδρία στην οποία ανατίθεται
1	1	11	5
2	4	12	4
3	6	13	4
4	6	14	6
5	2	15	1
6	4	16	4
7	5	17	5
8	4	18	5
9	2	19	3
10	3	20	2

Πίνακας 6.12: Αποτελέσματα ανάθεσης ιπτάμενων σε συνεδρίες για το πρόβλημα 2

Συνεδρία	Γλώσσα ανάθεσης
1	1 (FR)
2	0(EN)
3	1(FR)
4	0(EN)
5	1(FR)
6	0(EN)

Πίνακας 6.13: Αποτελέσματα ανάθεσης γλώσσας στις εκπαιδευτικές συνεδρίες του προβλήματος 2

Για balancelevel = 1

Λύνοντας το ίδιο πρόβλημα αλλά αυτή τη φορά με balancelevel 1 προστίθενται στην αντικειμενική συνάρτηση οι μεταβλητές ισορροπίας με δείκτες από 53 έως 132 και συντελεστή κόστους ίσο με 0,05.

Οι περιορισμοί ισορροπίας που προστίθενται παρουσιάζονται στο Παράρτημα Β λόγω του μεγάλου αριθμού τους

Σε αυτή την περίπτωση παρατηρούμε τις εξής αλλαγές σε σχέση με την περίπτωση όπου δεν υπήρχαν περιορισμοί ισορροπίας:

- Ο ιπτάμενος με αύξων αριθμό 6 ανατίθεται στην 6 συνεδρία, ενώ πριν ήταν στην συνεδρία 4
- Ο ιπτάμενος με αύξων αριθμό 8 ανατίθεται στην 2 συνεδρία, ενώ πριν ήταν στην συνεδρία 4
- Ο ιπτάμενος με αύξων αριθμό 17 ανατίθεται στην 1 συνεδρία, ενώ πριν ήταν στην συνεδρία 5
- Ο χρόνος επίλυσης του προβλήματος είναι 1.203sec
- Το πλήθος των μεταβλητών απόφασης είναι 132
- Η τιμή της αντικειμενικής συνάρτησης είναι 22,95

Παρατηρούμε πως η μετακίνηση και των τριών ιπτάμενων δεν έγινε εις βάρος των προτιμήσεων τους γιατί και πριν (balancelevel =0) είχαν τοποθετηθεί σε συνεδρίες για τις οποίες δεν είχαν εκφράσει επιθυμία, όπως ακριβώς και τώρα.

Για balancelevel =2

Λύνοντας το ίδιο πρόβλημα αλλά αυτή τη φορά με balancelevel 2προστίθενται στην αντικειμενική συνάρτηση οι μεταβλητές ισορροπίας με δείκτες από 53 έως 144 και συντελεστή κόστους ίσο με 0,05.

Οι περιορισμοί ισορροπίας που προστίθενται παρουσιάζονται στο Παράρτημα λόγω του μεγάλου αριθμού τους

Σε αυτή την περίπτωση παρατηρούμε τις εξής αλλαγές σε σχέση με την περίπτωση όπου δεν υπήρχαν περιορισμοί ισορροπίας:

- Ο ιπτάμενος με αύξων αριθμό 6 ανατίθεται στην 6 συνεδρία, ενώ πριν ήταν στην συνεδρία 4
- Ο ιπτάμενος με αύξων αριθμό 17 ανατίθεται στην 1 συνεδρία, ενώ πριν ήταν στην συνεδρία 5
- Ο χρόνος επίλυσης του προβλήματος είναι 1.453sec
- Το πλήθος των μεταβλητών απόφασης είναι 144
- Η τιμή της αντικειμενικής συνάρτησης είναι 23.35

Τα αποτελέσματα από την εκτέλεση του ίδιου προβλήματος για διαφορετικά επίπεδα ισορροπίας συνοψίζονται στον παρακάτω πίνακα.

balance level	**0**	**1**	**2**
Χρόνος επίλυσης	1.031sec	1.203sec	1.453sec
Σύνολο μεταβλητών απόφασης	52	132	144
Τιμή αντικειμενικής συνάρτησης	21	22,95	23,35

Πίνακας 6.14 : Σύγκριση αποτελεσμάτων από την επίλυση του προβλήματος 2 για διαφορετικά επίπεδα ισορροπία

6.3 Πειράματα

Μετά την επίλυση των δύο προβλημάτων διεξάγαμε μια σειρά πειραμάτων για διαφορετικά ζεύγη (N,S) αλλά και για διαφορετικά επίπεδα ισορροπίας και καταγράψαμε τους χρόνους επίλυσης αλλά και τον αριθμό των μεταβλητών απόφασης. Πιο συγκεκριμένα για κάθε ζεύγος (N, S) επιλύσαμε 10 διαφορετικά προβλήματα και το κάθε πρόβλημα το επιλύσαμε 3 φορές. Μιαγιαbalance level=0, μίαγιαbalance level=1 καιμίαγιαbalance level=2. Συνολικά λοιπόν για κάθε ζεύγος επιλύθηκαν 30 προβλήματα. Τα χαρακτηριστικά των ιπτάμενων κάθε φορά καθορίστηκαν με τη βοήθεια γεννήτριας τυχαιών αριθμών. Έτσι κάθε ιπτάμενος έχει:

Θέση

- Κυβερνήτη με πιθανότητα 0,5
- Συγκυβερνήτη με πιθανότητα 0,5

Γλώσσα

- Αγγλικά με πιθανότητα 0,33333
- Γαλλικά με πιθανότητα 0,33333
- Αγγλικά και Γαλλικά με πιθανότητα 0,333333

Λήξη εκπαιδευτικής συνεδρίας

- τον παρόντα μήνα X με πιθανότητα 0,33333
- τον επόμενο μήνα X+1 με πιθανότητα 0,333333
- το μεθεπόμενο μήνα X+2 με πιθανότητα 0,333333

Αρχαιότητα

ένας ακέραιος αριθμός, ο οποίος είναι διαφορετικός για κάθε ιπτάμενο.

Αριθμός εκπαιδευτικών συνεδριών

ένας ακέραιος αριθμός.που καθορίζεται από θετικό τυχαίο αριθμό μικρότερο της μονάδας, από το σύνολο των διαθέσιμων συνεδριών και από μία παράμετρο η οποία είναι ίση με 0,4

Τιμή προτεραιότητας κάθε εφικτής συνεδρίας

- 0 με πιθανότητα 0,6
- θετική τιμή με πιθανότητα 0,4

Κάθε διαθέσιμη εκπαιδευτική συνεδρία έχει:

Συνολική χωρητικότητα θέσεων: 20 άτομα

Χωρητικότητα πληρώματος καμπίνας :10 άτομα

χωρητικότητα πληρώματος πιλοτηρίου :14 άτομα

Χωρητικότητα κυβερνητών:8 άτομα

Χωρητικότητα συγκυβερνητών:8 άτομα

Ο πίνακας 6.15 παρουσιάζει το μέσο χρόνο επίλυσης (σε δευτερόλεπτα) των δέκα προβλημάτων για κάθε ζεύγος (N,S) και για κάθε balancelevel

balance level	0	1	2
(50,5)	0,9953	1,2546	1,217
(100,10)	1,4342	2,4438	2,781
(150,15)	1,7144	3,0943	3,6092
(200,20)	2,5544	8,7196	9,8561
(250,25)	3,7963	21,5542	47,4524

Πίνακας 6.15 : Μέσος χρόνος επίλυσης των προβλημάτων για διαφορετικά ζεύγη (N,S) και για διαφορετικά επίπεδα ισορροπίας

Ενώ ο πίνακας 6,16 παρουσιάζει το μέσο αριθμό μεταβλητών των δέκα προβλημάτων για κάθε ζεύγος (N,S) και για κάθε balancelevel

balance level	0	1	2
(50,5)	110	170	190
(100,10)	352,5	622,5	712,5
(150,15)	708,9	1338,9	1548,9
(200,20)	1225,2	2365,2	2745,2
(250,25)	2033,1	3833,1	4433,1

Πίνακας 6.16 : Μέσος αριθμός μεταβλητών των προβλημάτων για διαφορετικά ζεύγη (N,S) και για διαφορετικά επίπεδα ισορροπίας

Από τα αποτελέσματα που περιέχονται στους πίνακες γίνεται φανερό ότι για το ίδιο ζεύγος (N,S) όσο αυξάνουμε το επίπεδο ισορροπίας αυξάνεται ο αριθμός των μεταβλητών και συνεπακόλουθα ο χρόνος επίλυσης του προβλήματος. Η άυξηση αυτή είναι μικρή για μικρό αριθμό ιπταμένων και συνεδριών αλλά γίνεται πιο αισθητή όταν τα σύνολα N και S παίρνουν μεγάλες τιμές.

ΕΠΙΛΟΓΟΣ

Αυτό η ενότητα της εργασίας αποτελείται από δύο κεφάλαια. Στο πρώτο κάνουμε μια ανασκόπηση της εργασίας και παρουσιάζουμε τα συμπεράσματα στα οποία καταλήξαμε, ενώ στο δεύτερο κεφάλαιο προτείνουμε πιθανά σημεία στα οποία θα μπορούσαν να επικεντρωθούν μελλοντικές εργασίες.

7. Σύνοψη και συμπεράσματα

Σκοπός αυτής της εργασίας είναι η ανάπτυξη και εκτέλεση κώδικα για την εύρεση βέλτιστης λύσης στο πρόβλημα ανάθεσης του ιπτάμενου προσωπικού προς εκπαίδευση με στόχο τη μείωση του προσωπικού που δεν ανατίθεται σε εκπαιδευτικές συνεδρίες. Για την ανάπτυξη του κώδικα μελετήσαμε έρευνες και ανάλογους κώδικες που έχουν αναπτυχθεί για τον ίδιο σκοπό και στη συνέχεια της εργασίας περιγράψαμε διεξοδικά τις συνιστώσες του προβλήματος αλλά και τις σχέσεις μεταξύ αυτών. Επόμενο βήμα της μελέτης μας ήταν η ποσοτικοποίηση αυτών των σχέσεων και η κατασκευή του μαθηματικού μοντέλου. Στηριζόμενοι σε αυτό το μοντέλο υλοποιήσαμε τον κώδικά μας σε γλώσσα προγραμματισμού Cκαι το επιλύσαμε με το λογισμικό βελτιστοποίησης προβλημάτων CPLEX. Για να εξετάσουμε την ορθότητα του μοντέλου μας επιλύσαμε μικρά και διαχειρίσιμα προβλήματα προκειμένου να διαπιστώσουμε όχι αν η λύση είναι η βέλτιστη αλλά για να διαπιστώσουμε αν ο κώδικας αναπαράγει τους σωστούς περιορισμούς και η λύση που προκύπτει είναι λογική. Αφού σιγουρευτήκαμε ότι ο κώδικας αναπαράγει τους σωστούς περιορισμούς και τις σωστές μεταβλητές απόφασης,

Στη συνέχεια εκτελέσαμε μια σειρά πειραμάτων προκειμένου να εξετάσουμε τους χρόνους που απαιτούνται για την επίλυση προβλημάτων διαφορετικού μεγέθους. Όπως είναι φυσικό διαπιστώνουμε πως όσο μεγαλύτερο είναι το πλήθος των πιλότων και των συνεδριών τόσο περισσότερος χρόνος απαιτείται για την επίλυση του προβλήματος. Μάλιστα όταν θέτουμε επίπεδο ισορροπίας μεγαλύτερο του μηδενός απαιτείται ακόμα μεγαλύτερος χρόνος επίλυσης.

8. Μελλοντικές επεκτάσεις

Ολοκληρώνοντας την εργασία αυτή προτείνουμε ορισμένα θέματα τα οποία θα μπορούσαν να αποτελέσουν έμπνευση για μετέπειτα έρευνα, χωρίς αυτό να σημαίνει πως δεν μπορεί να υπάρχουν και άλλα θέματα προς εξέταση.

Μια επέκταση του μοντέλου με την οποία θα μπορούσε να ασχοληθεί κανείς είναι η βελτιστοποίηση του προγράμματος εκπαίδευσης του ιπτάμενου προσωπικού με χρονικό ορίζοντα όχι τον ένα μήνα αλλά τους τρεις μήνες. Επίσης ένα ακόμα πεδίο έρευνας θα μπορούσε να ασχοληθεί με τη βέλτιστη ανάθεση σε συνεδρίες όχι μόνο των εκπαιδευόμενων αλλά και των εκπαιδευτών. Θα μπορούσε λοιπόν να κατασκευαστεί ένα μαθηματικό μοντέλο για την ταυτόχρονη βελτιστοποίηση δύο κριτηρίων(βελτιστοποίηση με πολλαπλά κριτήρια). Τοέναθα αφορά την ελαχιστοποίηση του αριθμού των εκπαιδευόμενων που μένουν εκτός εκπαίδευσης τον τρέχοντα μήνα και ο αλλός την ελαχιστοποίηση του αριθμού των εκπαιδευτών που δεν συμμετέχουν στις συνεδρίες του μήνα. Και σε αυτό το μοντέλο θα υπήρχαν περιορισμοί ανάθεσης, περιορισμοί αρχαιότητας, περιορισμοί χωρητικότητας και γλώσσας.

ΠΑΡΑΡΤΗΜΑ Α’

Στο Παράρτημα Α’ παρουσιάζεται ο κώδικας που αναπτύχθηκε και που χρησιμοποιήθηκε για την επίλυση των δύο προβλημάτων ανάθεσης ιπταμένων για εκπαίδευση του 6^{ου} κεφαλαίου αλλά και για την διεξαγωγή τον πειραμάτων.

```
#include<ilcplex/ilocplex.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "math.h"

#define  N   15 // number of trainees
#define  S   4// number of training sessions

CPXENVptr     env = NULL;
CPXLPptr      lp = NULL;
int          Not_Expire;    // number of id's that do not expire in X
int          tot_columns;   // number of (assignment+non-assignment) decision vars
int          tot_valid_bids;// number of total valid bids for one id
double       *obj, *lb, *ub;
double       rhs;
double       objective =0;
double       total_seats = 0;
int          rmatbeg = 0;
char         sense;
int          *rmatind;
double       *rmatval;
char         *ctype;
char         *vartype;
char         *row_name;
int          VarIndex[N][S+1];
int               Max_Priority[N];
```

```
typedef struct session_tag {
   int  index;
   double  total_capacity;
   double capacity_in_cockpit;
   double capacity_in_cabin;
   double capacity_CP;
   double capacity_FO;
   int  priority;
   int  language;  // 0 = not defined, 1 = french, 2 = english
   session_tag *next_session;
} session;

typedef struct pilot_tag {
   int  expiration; // 0 = X, 1 = X+1, 2 = X+2
   int  position;   // 1 = CP, 2 = FO
   int  seniority;  // the smaller number, the more senior
   int  language;   // 0 = speaks both, 1 = french, 2 = english
   int  pilotindex;
   int  num_ofvalidbids;
   session_tag *valid;
   pilot_tag *next_pilot;
} pilot;

pilot_tag   *PilotList;
session_tag *SessionList;

typedef struct Coeff_tag {
   int  Sen;      // seniority
   int  Prior;    // priority
   char RecType;  // 'M' for empty seats variable, 'P' for assignment & non-assignment
variables
   int  lLogIdx;
   Coeff_tag *RightLink;
   Coeff_tag *LeftLink;
} Coeff;
```

```
Coeff_tag *PenaltyList, *PList;

/********** Adds to PenaltyList **********/
// adds pilots data including the preferences into a list
// sorts the penalty nodes, so that more junior ids go to the front
// higher priority nodes of the same id go to the front, and the unique type 'M' node goes to the end of the list

void AddToPenaltyList (int iLogIdx, int sen, int max_priori, int prior, char recType, Coeff_tag **L)
{ // inserts at right position
  Coeff_tag *Node, *K, *R;

  Node = (Coeff_tag *)malloc(sizeof(Coeff_tag));
  Node->Sen = sen;
  Node->Prior = prior;
  Node->RecType = recType;
  Node->ILogIdx = iLogIdx;
  Node->RightLink = NULL;
  Node->LeftLink = NULL;

  if ((*L) == NULL) {
    (*L) = Node;
  }
  else if (recType == 'M') { // node of empty seats
            K = (*L);
            R = K->RightLink;
            while (R != NULL) {
                    K = K->RightLink;
        R = R->RightLink;
            }
            K->RightLink = Node;
     Node->RightLink = NULL;
            Node->LeftLink = K;
  } //  end else if
  else if ((Node->Sen > (*L)->Sen) || ((Node->Sen == (*L)->Sen) && (Node->Prior > (*L)->Prior))){
```

```
            Node->RightLink = (*L);
            (*L)->LeftLink = Node;
        (*L) = Node;
    }
    else {
            K = (*L);
            R = K->RightLink;
            while (R != NULL)
                    if ((R->RecType == 'P') && (R->Sen > sen )){
                            K = K->RightLink;
                       R = R->RightLink;
                    }
                    else break;
               if (R == NULL){
                            K->RightLink = Node;
                       Node->LeftLink = K;
                       Node->RightLink = NULL;
                    }
               else if (R->Sen == sen){
                            while (R != NULL)
                                    if ((R->Sen == sen) && (R->Prior > prior )){
                                            K = K->RightLink;
                            R = R->RightLink;
                                    }
                                    else break;
                                    K->RightLink = Node;
                          Node->RightLink = R;
                          Node->LeftLink = K;
                          if (R != NULL) R->LeftLink = Node;
                    }// end if (R->Sen == sen)
               else{
                            K->RightLink = Node;
                       Node->RightLink = R;
                       Node->LeftLink = K;
                            if (R != NULL) R->LeftLink = Node;
                    }// end else
    }// end else
```

```
}

/*****  initializes the cplex environment  *****/
int init_problem()
{
   int status;

   env = CPXopenCPLEX (&status);
   if (status != 0) printf("Cannot open env\n");

   lp = CPXcreateprob (env, &status, "model");
   if (status != 0) printf("Can not create prob\n");

   return status;
}

/***************  data input  ******************/
int fill_lists()
{
   int  ii, kk, num_valid;
   int  pref, chosen, visited;
   int  sen[N];
   double  random_number, Seed;
   pilot_tag  *new_pilot,*temp1;
   session_tag *new_session, *temp2;

   srand(time(0));
   Seed = rand();
   printf("seed=%lf\n",Seed);
   //Seed =8961;
   //srand(time(0));
   srand(Seed);
   for(ii = 0; ii <= N-1; ii++)
     Max_Priority[ii] = -1;
     sen[0] = 1; // seniority calculation
     for (ii = 1; ii <= N-1; ii++){
       random_number = (double)rand()/(double)RAND_MAX;
```

```
        if (random_number < 0.5){
          for (kk = ii-1; kk >= 0; kk--)
          sen[kk+1] = sen[kk];
                sen[0] = ii+1;
  }
  else sen[ii] = ii+1;
} // end for loop
for (ii = 0; ii <= S-1; ii++){ // data input for each training session
  new_session = (session_tag*) malloc(sizeof(session_tag));
  new_session->index = ii+1;
        new_session->priority = -1;
        new_session->language = 0;
        //new_session->total_capacity = (int)(8*rand()/double(RAND_MAX+1))+3;
        new_session->total_capacity = 20;
        total_seats=total_seats+new_session->total_capacity;
        new_session->capacity_in_cockpit=14;
        new_session->capacity_CP=8;
        new_session->capacity_FO=8;
        new_session->capacity_in_cabin=10;
        new_session->next_session = NULL;
  if (SessionList == NULL)
          SessionList = new_session;
  else{
          temp2 = SessionList;
        while (temp2->next_session != NULL)
    temp2 = temp2->next_session;
          temp2->next_session = new_session;
  }
      }// end for loop
for (ii = 0; ii <= N-1; ii++){ // data input for each pilot
   new_pilot = (pilot_tag*) malloc(sizeof(pilot_tag));
         random_number = (double)rand()/(double)RAND_MAX; // position
         if (random_number <= 0.3333333333)
           new_pilot->position = 1;
         else if (random_number <= 0.66666666)
           new_pilot->position = 2;
         else
```

```
        new_pilot->position = 3;

   random_number = (double)rand()/(double)RAND_MAX;//expiration
      if (new_pilot->position != 3) {
       if (random_number <= 0.3333333)
         new_pilot->expiration = 0 ;
       else if (random_number <= 0.6666666)
         new_pilot-> expiration = 1 ;
       else
                  new_pilot->expiration = 2;
            }
      else
            new_pilot->expiration =0;

          new_pilot->seniority = sen[ii]; // seniority
new_pilot->valid = NULL;
pref = 1;
      random_number = (double)rand()/(double)RAND_MAX; // num of valid bids
      num_valid = (int)ceil(random_number*S*0.4);
      chosen = 0;  visited = 0;
for (kk = 1; kk <= S; kk++){
new_session = (session_tag*) malloc(sizeof(session_tag));
random_number = (double)rand()/(double)RAND_MAX;
      if (random_number <= ((double)(num_valid-chosen)/(double)(S-visited))){
        new_session->index = kk;
            chosen++;
        random_number = (double)rand()/(double)RAND_MAX;
        if (random_number <= 0.6) {
                  new_session->priority = 0;
            }

        else {
              new_session->priority = pref;
            pref++;
            }
        new_session->next_session = NULL;
  if (new_pilot->valid == NULL)
```

```
                    new_pilot->valid = new_session;
                else{
                    temp2 = new_pilot->valid;
                    while (temp2->next_session != NULL)
                      temp2 = temp2->next_session;
                      temp2->next_session = new_session;
                }
            } // end  if random loop
            visited++;
                if (chosen >= num_valid) break;
        } // end for kk loop

        new_pilot->num_ofvalidbids = num_valid;
            if ((new_pilot->expiration==0) && (new_pilot->num_ofvalidbids == 1)){
                    new_pilot->language =0;
            }
            else {
        random_number = (double)rand()/(double)RAND_MAX; // language
        if (random_number <= 0.333333)
          new_pilot->language = 0;
        else if (random_number <= 0.666666)
                new_pilot->language = 1;
        else
                new_pilot->language = 2;
            }
        new_pilot->pilotindex = ii+1;
        new_pilot->next_pilot = NULL;
        Max_Priority[ii] = pref-1;
        if (PilotList == NULL)
          PilotList = new_pilot;
        else{
                temp1 = PilotList;
                while (temp1->next_pilot != NULL)
                  temp1 = temp1->next_pilot;
            temp1->next_pilot = new_pilot;
        }
    }// end for ii loop
```

```
   return 0;
}

/***************  data input for first example ******************/
/*int fill_lists()
{
   int  ii, kk, num_valid;
   int  sen[N], pos[N], exp[N], maxpr[N];
   int  lang[N], num_val[N], val_bids[N][S], ses_pref[N][S];
   int  Lang[S], cap[S];
   pilot_tag  *new_pilot,*temp1;
   session_tag *new_session,*temp2;

pos[0]=1; pos[1]=3; pos[2]=2; pos[3]=2; pos[4]=1;
   pos[5]=3; pos[6]=2; pos[7]=3; pos[8]=3; pos[9]=2;
   pos[10]=3;pos[11]=3;pos[12]=3;pos[13]=1; pos[14]=1;

   lang[0]=0; lang[1]=0; lang[2]=0; lang[3]=1; lang[4]=2;
   lang[5]=0; lang[6]=1; lang[7]=2; lang[8]=0; lang[9]=0;
   lang[10]=0; lang[11]=0; lang[12]=1;lang[13]=1;lang[14]=2;

   sen[0]=12;  sen[1]=8; sen[2]=7; sen[3]=4; sen[4]=2;
   sen[5]=1;  sen[6]=3; sen[7]=5;  sen[8]=6; sen[9]=9;
   sen[10]=10; sen[11]=11; sen[12]=13; sen[13]=14; sen[14]=15;

   exp[0]=0; exp[1]=0; exp[2]=1; exp[3]=2; exp[4]=1;
   exp[5]=0; exp[6]=0; exp[7]=0; exp[8]=0; exp[9]=2;
   exp[10]=0; exp[11]=0; exp[12]=0; exp[15]=1; exp[14]=1;

   num_val[0]=1;num_val[1]=1; num_val[2]=2; num_val[3]=2;
num_val[4]=1;
   num_val[5]=2; num_val[6]=2; num_val[7]=2; num_val[8]=2;
num_val[9]=2; num_val[10]=1; num_val[11]=1; num_val[12]=2;
num_val[13]=2;num_val[14]=2;

val_bids[0][0] = 2;  val_bids[1][0] = 1;  val_bids[2][0] = 2;
val_bids[2][1] = 4;  val_bids[3][0] = 2;  val_bids[3][1] = 4;
```

```
val_bids[4][0] = 2;
val_bids[5][0] = 1;  val_bids[5][1] = 2;  val_bids[6][0] = 1;
val_bids[6][1] = 3; val_bids[7][0] = 1;
  val_bids[7][1] = 3;val_bids[8][0] = 1;  val_bids[8][1] = 4;
val_bids[9][0] = 3;
  val_bids[9][1] = 4;  val_bids[10][0] = 2;  val_bids[11][0] = 1;
val_bids[12][0] = 2;
  val_bids[12][1] = 3;  val_bids[13][0] = 1;  val_bids[13][1] = 4;
  val_bids[14][0] = 1;  val_bids[14][1] = 3;

  ses_pref[0][0] = 0;  ses_pref[1][0] = 0;  ses_pref[2][0] = 0;
  ses_pref[2][1] = 0;  ses_pref[3][0] = 1;  ses_pref[3][1] = 2;
ses_pref[4][0] = 1;
  ses_pref[5][0] = 0;  ses_pref[5][1] = 1;  ses_pref[6][0] = 0;
ses_pref[6][1] = 0; ses_pref[7][0] = 0;
  ses_pref[7][1] = 0;  ses_pref[8][0] = 1;  ses_pref[8][1] = 0;
ses_pref[9][0] = 0;
  ses_pref[9][1] = 0;  ses_pref[10][0] = 1;  ses_pref[11][0] = 1;
ses_pref[12][0] = 0;
  ses_pref[12][1] = 1;  ses_pref[13][0] = 0;  ses_pref[13][1] = 1;
  ses_pref[14][0] = 1;  ses_pref[14][1] = 0;

  maxpr[0]=0; maxpr[1]=0; maxpr[2]=0; maxpr[3]=2; maxpr[4]=1;
  maxpr[5]=1; maxpr[6]=0; maxpr[7]=0; maxpr[8]=1; maxpr[9]=0;
  maxpr[10]=1; maxpr[11]=1; maxpr[12]=1; maxpr[13]=1;maxpr[14]=1;

  // sessions
Lang[0]=0; Lang[1]=0; Lang[2]=0; Lang[3]=0;

  for (ii = 0; ii <= S-1; ii++){ // data input for each training session
new_session = (session_tag*) malloc(sizeof(session_tag));

        new_session->index = ii+1;
        new_session->priority = -1;
        new_session->language = Lang[ii];
        new_session->total_capacity=20;
        total_seats=total_seats+new_session->total_capacity;
```

```
            new_session->capacity_in_cockpit=14;
            new_session->capacity_in_cabin=10;
            new_session->capacity_CP=8;
            new_session->capacity_FO=8;
            new_session->next_session = NULL;

        if (SessionList == NULL)
            SessionList = new_session;
        else{
             temp2 = SessionList;
               while (temp2->next_session != NULL)
          temp2 = temp2->next_session;
                 temp2->next_session = new_session;
         }
      } // end for loop

      for (ii = 0; ii <= N-1; ii++){ // data input for each pilot
        new_pilot = (pilot_tag*) malloc(sizeof(pilot_tag));
            new_pilot->expiration = exp[ii];
            new_pilot->position = pos[ii];
        new_pilot->seniority = sen[ii]; // seniority
        new_pilot->valid = NULL;

            num_valid = num_val[ii];
            for (kk = 0; kk <= num_val[ii]-1; kk++){
    new_session = (session_tag*) malloc(sizeof(session_tag));
            new_session->index = val_bids[ii][kk];
            new_session->priority = ses_pref[ii][kk];
            new_session->next_session = NULL;
             if (new_pilot->valid == NULL)
                        new_pilot->valid = new_session;
                  else{
                        temp2 = new_pilot->valid;
                        while (temp2->next_session != NULL)
                          temp2 = temp2->next_session;
                          temp2->next_session = new_session;
                  }
```

```
            } // end for kk loop

            new_pilot->num_ofvalidbids = num_val[ii];
            new_pilot->language = lang[ii];
            new_pilot->pilotindex = ii+1;
            new_pilot->next_pilot = NULL;

            Max_Priority[ii] = maxpr[ii];
       if (PilotList == NULL)
         PilotList = new_pilot;
       else{
              temp1 = PilotList;
              while (temp1->next_pilot != NULL)
                temp1 = temp1->next_pilot;
          temp1->next_pilot = new_pilot;
       }
     } // end for ii loop

   return 0;
}
*/


/***********  prints problem's data  *************/
int print_lists()
{
   pilot_tag  *pltptr;
   session_tag *sesptr;

   FILE *file;
   file = fopen("file.txt","w+");

   pltptr = PilotList;
   while (pltptr != NULL) {
    fprintf(file,"\nPrinting pilot with index = %d:\n", pltptr->pilotindex);
          fprintf(file,"Expiration = %d\n", pltptr->expiration);
```

```
            fprintf(file,"Position = %d\n", pltptr->position);
            fprintf(file,"Seniority = %d\n", pltptr->seniority);
            fprintf(file,"Language = %d\n", pltptr->language);
            fprintf(file,"Num of valid Bids = %d\n", pltptr->num_ofvalidbids);
            fprintf(file,"Valid sessions of pilot %d:\n", pltptr->pilotindex);

            sesptr = pltptr->valid;
            while(sesptr != NULL){
      fprintf(file,"\tSession index = %d, priority = %d\n", sesptr->index, sesptr->priority);
       sesptr = sesptr->next_session;
            }
            pltptr = pltptr->next_pilot;
   } // end while (pltptr != NULL)

   sesptr = SessionList;
   while (sesptr != NULL) {
     fprintf(file,"\nPrinting session with index = %d:\n", sesptr->index);
            fprintf(file,"Language = %d\n", sesptr->language);
            fprintf(file,"Total capacity = %lf\n", sesptr->total_capacity);
            fprintf(file,"capacity_in_cockpit= %lf\n",sesptr->capacity_in_cockpit);
            fprintf(file,"capacity in cabin=  %lf\n",sesptr->capacity_in_cabin);
            fprintf(file,"capacity of CP's=  %lf\n",sesptr->capacity_CP);
            fprintf(file,"capacity of FO's=  %lf\n",sesptr->capacity_FO);
            sesptr = sesptr->next_session;
   } // end while (sesptr != NULL)
   fprintf(file,"total available seats= %lf\n",total_seats);
   fclose(file);
   return 0;
}

/***********  memory allocation *********/
void  getmem_newcols(double  **obj_ptr,  double  **lb_ptr,  double  **ub_ptr,  char **ctype_ptr, int size)
{
   *obj_ptr    = (double *)malloc(size * sizeof(double));
   *lb_ptr     = (double *)malloc(size * sizeof(double));
```

```
  *ub_ptr     = (double *)malloc(size * sizeof(double));
  *ctype_ptr  = (char *)malloc(size * sizeof(char));
}

void freemem_newcols(double **obj_ptr, double **lb_ptr, double **ub_ptr, char
**ctype_ptr)
{
  free(*obj_ptr);
  free(*lb_ptr);
  free(*ub_ptr);
  free(*ctype_ptr);
}

void getmem_addrows(int **rmatind_ptr, double **rmatval_ptr, int size)
{
  *rmatind_ptr  = (int *)malloc(size * sizeof(int));
  *rmatval_ptr  = (double *)malloc(size * sizeof(double));
}

void freemem_addrows(int **rmatind_ptr, double **rmatval_ptr)
{
  if (*rmatind_ptr != NULL)
        free(*rmatind_ptr);
  if (*rmatval_ptr != NULL)
        free(*rmatval_ptr);
}

/****** adds the decision variables (columns) of the problem ******/
int add_columns()
{
  int status;
  int jj;

  pilot_tag  *pltptr;
  session_tag *sesptr;

  getmem_newcols(&obj, &lb, &ub, &ctype, 1);
```

```
obj[0] = 0;
lb[0] = 0;
ub[0] = 1;
ctype[0] = 'B';

FILE *file;
file = fopen("file.txt","a+");
PenaltyList=NULL;

tot_valid_bids = 0;
pltptr = PilotList;
while (pltptr != NULL) {
  sesptr = pltptr->valid;
        while(sesptr != NULL){
    status = CPXnewcols(env, lp, 1, obj, lb, ub, ctype, NULL);
    if (status != 0){
                printf("Failed to add assignment var, exiting\n");
           return status;
    }

        VarIndex[pltptr->pilotindex-1][sesptr->index-1] = CPXgetnumcols(env,lp)-1;
        tot_valid_bids++;
        if (sesptr->priority == 0)
                AddToPenaltyList(tot_valid_bids-1, pltptr->seniority,Max_Priority[pltptr->pilotindex -1],0,'P',&PenaltyList); // if has not bid for this session , add with priority P
        else
                AddToPenaltyList(tot_valid_bids-1, pltptr->seniority,Max_Priority[pltptr->pilotindex -1],Max_Priority[pltptr->pilotindex -1] - sesptr->priority+2,'P',&PenaltyList); //if has bid for this session */

        /*PList=PenaltyList;
         fprintf(file,"*******Start of Penalty List******\n");
              while (PList!=NULL){
         fprintf(file,"ILogIdx=%d\n",PList->ILogIdx);
              fprintf(file,"Sen=%d\n",PList->Sen);
              fprintf(file,"Prior=%d\n",PList->Prior);
```

```
                    fprintf(file,"RecType=%c\n\n",PList->RecType);
                    PList=PList->RightLink;
                    }
                    fprintf(file,"*******End of Penalty List******\n");
                    getchar();*/

                    sesptr = sesptr->next_session;
            }// end while(sesptr != NULL)
            if (pltptr->expiration > 0) Not_Expire++;
            pltptr = pltptr->next_pilot;
    }// end while (pltptr != NULL)
    tot_columns = tot_valid_bids + Not_Expire;

    jj = 0;
    pltptr = PilotList;
    while (pltptr != NULL) {
      if (pltptr->expiration > 0) {
              AddToPenaltyList(tot_valid_bids+jj,      pltptr->seniority,Max_Priority[pltptr-
>pilotindex -1],1,'P',&PenaltyList);
             /* PList=PenaltyList;
              fprintf(file,"*******Start of Penalty List******\n");
              while (PList!=NULL){
                fprintf(file,"ILogIdx=%d\n",PList->ILogIdx);
                fprintf(file,"Sen=%d\n",PList->Sen);
                fprintf(file,"Prior=%d\n",PList->Prior);
                fprintf(file,"RecType=%c\n\n",PList->RecType);
                PList=PList->RightLink;
              }
              fprintf(file,"*******End of Penalty List******\n");
              getchar();*/

             jj++;
        status = CPXnewcols(env, lp, 1, obj, lb, ub, ctype, NULL);
        if (status != 0){
              printf("Failed to add non-assignment var, exiting\n");
              return status;
        }
```

```
            VarIndex[pltptr->pilotindex-1][S] = CPXgetnumcols(env,lp)-1;
          } // end if (pltptr->expiration > 0)
        pltptr = pltptr->next_pilot;
  } // end while (pltptr != NULL)

  freemem_newcols(&obj, &lb, &ub, &ctype);
  fclose(file);
  return status;
}

/***********  adds assignments constraints  **********/
// adds the constraints that ensure that each ID will be assigned to a session
int add_assignments()
{
  int  cnt, not_exp_cnt, nzcnt;
  int  status;
  pilot_tag  *pltptr;
  session_tag *sesptr;

  sense = 'E';
  rhs = 1;
  cnt=0;        // index of the corresponding decision variable
  not_exp_cnt=0; // counts the num of id's that do not expire in x

  FILE *file;
  file = fopen("file.txt","a+");

  getmem_addrows(&rmatind, &rmatval, S+1);
  row_name = (char*) malloc(10*sizeof(char));

  pltptr = PilotList;
  while (pltptr != NULL) {
    sesptr = pltptr->valid;
          nzcnt=0;    // index in the arrays rmatind and rmatval
          while(sesptr != NULL){
            rmatind[nzcnt]=cnt;
            rmatval[nzcnt]=1;
```

```
            cnt++;
            nzcnt++;
            sesptr = sesptr->next_session;
          }// end while(sesptr != NULL)
          if (pltptr->expiration > 0) {
            rmatind[nzcnt]=tot_valid_bids+not_exp_cnt;
            rmatval[nzcnt]=1;
            not_exp_cnt++;
                nzcnt++;
          }
          if (nzcnt <= 0){
            pltptr = pltptr->next_pilot;
            continue;
          }

    sprintf(row_name, "MC%d", pltptr->pilotindex);
    status = CPXaddrows(env, lp, 0, 1, nzcnt, &rhs, &sense,&rmatbeg, rmatind,
rmatval, NULL, &row_name);
    if (status != 0){
                 printf("Failed to add assignment constraint for pilot%d, exiting\n",
pltptr->pilotindex);
             return status;
          }
          pltptr = pltptr->next_pilot;
  } // end while (pltptr != NULL)
  freemem_addrows(&rmatind, &rmatval);
  free(row_name);
  fclose(file);
  return status;
}

/*********  adds  capacity constraints for each session *********/
// adds the constraints that ensure that the total number of trainees that will be
assigned to each session will not exceed the total capacityof this session
int add_capacity_trainees()
{
  int  cnt, nzcnt;
```

```
  int  status;
  pilot_tag  *pltptr;
  session_tag *sesptr1, *sesptr2;
  sense = 'L';
  getmem_addrows(&rmatind, &rmatval, N);
  row_name = (char*) malloc(32*sizeof(char));

  sesptr1 = SessionList;
  while (sesptr1 != NULL){
    cnt = 0;   // index of corresponding decision variable
          nzcnt = 0; // index in the arrays rmatind and rmatval
          pltptr = PilotList;
          while (pltptr != NULL) {
            sesptr2 = pltptr->valid;
                  while (sesptr2 != NULL){
                    if (sesptr1->index == sesptr2->index) {
                      rmatind[nzcnt] = cnt;
                      rmatval[nzcnt] = 1;
                      nzcnt++;
                    }
                    cnt++;
                    sesptr2 = sesptr2->next_session;
                  } // end while(sesptr2 != NULL)
                  pltptr = pltptr->next_pilot;
          }// end while (pltptr != NULL)
          if (nzcnt <= 0) {
            sesptr1 = sesptr1->next_session;
                  continue;
          }
    sprintf(row_name,"TOTALCAP_session_%d", sesptr1->index);
          status = CPXaddrows(env, lp, 0, 1, nzcnt, &sesptr1->total_capacity, &sense,
&rmatbeg, rmatind, rmatval, NULL, &row_name);
          if (status != 0){
            printf("Failed to add capacity constraint for session %d, exiting\n", sesptr1-
>index);
                  return status;
          }
```

```
            sesptr1 = sesptr1->next_session;
   } //while(sesptr1 != NULL)
   freemem_addrows (&rmatind, &rmatval);
   free (row_name);
   return status;
}

/*********  adds  capacity constraints for cockpit and cabin *********/
// adds the constraints that ensure that the capacity in the cockpit and in the cabin will
no be violated
int add_capacity_cockpit_cabin()
{
   int  cnt, nzcnt;
   int  status,times;
   pilot_tag  *pltptr;
   session_tag *sesptr1, *sesptr2;

   sense = 'L';
   FILE *file;
   file = fopen("file.txt","a+");
   times=1;
   getmem_addrows(&rmatind, &rmatval, N);
   row_name = (char*) malloc(15*sizeof(char));
   for (times=1; times<=2; times++) {
            sesptr1 = SessionList;
            while (sesptr1 != NULL){
        cnt = 0;   // index of corresponding decision variable
              nzcnt = 0; // index in the arrays rmatind and rmatval
              pltptr = PilotList;
              while (pltptr != NULL) {
                    if ((times==1) && (pltptr->position > 2)){
                          cnt = cnt+pltptr->num_ofvalidbids;
                          pltptr=pltptr->next_pilot;
                          continue;
                       }
                if ((times==2) && (pltptr->position <=2)){
                          cnt = cnt+pltptr->num_ofvalidbids;
```

```
                        pltptr=pltptr->next_pilot;
                        continue;
                    }
            sesptr2 = pltptr->valid;
                while (sesptr2 != NULL){
              if (sesptr1->index == sesptr2->index) {
                    rmatind[nzcnt] = cnt;
                    rmatval[nzcnt] = 1;
                    nzcnt++;
                        }
                  cnt++;
                  sesptr2 = sesptr2->next_session;
                  } // end while(sesptr2 != NULL)
                pltptr = pltptr->next_pilot;
               }// end while (pltptr != NULL)
      if (nzcnt <= 0) {
            sesptr1 = sesptr1->next_session;
                continue;
               }
          if (times==1) sprintf(row_name, "COCKPIT_ses_%d", sesptr1->index);
          else sprintf(row_name, "Cabin_ses_%d", sesptr1->index);
      if (times==1) rhs=sesptr1->capacity_in_cockpit;
          else rhs=sesptr1->capacity_in_cabin;
      status = CPXaddrows(env, lp, 0, 1, nzcnt, &rhs, &sense, &rmatbeg, rmatind,
rmatval, NULL, &row_name);
          if (status != 0){
               printf("Failed to add capacity constraint for session %d, exiting\n",
sesptr1->index);
               return status;
               }
          sesptr1 = sesptr1->next_session;
        } //while(sesptr1 != NULL)
  }
  fclose(file);
  freemem_addrows (&rmatind, &rmatval);
  free (row_name);
  return status;
```

```
}

/*********  adds  capacity constraints for FO's and CP's *********/
// adds the constraints that ensure that the capacity of CP's and FO's will not be violated
int add_capacity_per_position()
{
  int  cnt, nzcnt;
  int  status,times;
  pilot_tag  *pltptr;
  session_tag *sesptr1, *sesptr2;

  sense = 'L';
  FILE *file;
  file = fopen("file.txt","a+");
  times=1;
  getmem_addrows(&rmatind, &rmatval, N);
  row_name = (char*) malloc(15*sizeof(char));
  for (times = 1; times <= 2; times++) {//1 for CP's and 2 for FO's
    sesptr1 = SessionList;
          while (sesptr1 != NULL){
      cnt = 0;   // index of corresponding decision variable
            nzcnt = 0; // index in the arrays rmatind and rmatval
            pltptr = PilotList;
            while (pltptr != NULL) {
                  if ((times==1) && (pltptr->position != 1)){
                        cnt = cnt+pltptr->num_ofvalidbids;
                        pltptr=pltptr->next_pilot;
                        continue;
                  }
              if ((times==2) && (pltptr->position != 2)){
                        cnt = cnt+pltptr->num_ofvalidbids;
                        pltptr=pltptr->next_pilot;
                        continue;
                  }
              sesptr2 = pltptr->valid;
                  while (sesptr2 != NULL){
```

```
                if (sesptr1->index == sesptr2->index) {
                        rmatind[nzcnt] = cnt;
                        rmatval[nzcnt] = 1;
                        nzcnt++;
                            }
                    cnt++;
                    sesptr2 = sesptr2->next_session;
                       }// end while(sesptr2 != NULL)
                  pltptr = pltptr->next_pilot;
               }// end while (pltptr != NULL)
      if (nzcnt <= 0) {
            sesptr1 = sesptr1->next_session;
                continue;
               }
          if (times==1) sprintf(row_name, "CP'S_ses_%d", sesptr1->index);
          else sprintf(row_name, "FO's_ses_%d", sesptr1->index);
      if (times==1)
                        rhs=sesptr1->capacity_CP;
          else
                        rhs=sesptr1->capacity_FO;

      status = CPXaddrows(env, lp, 0, 1, nzcnt, &rhs, &sense, &rmatbeg, rmatind,
rmatval, NULL, &row_name);
          if (status != 0){
                printf("Failed to add capacity constraint for session %d, exiting\n",
sesptr1->index);
                return status;
               }
          sesptr1 = sesptr1->next_session;
        } //while(sesptr1 != NULL)
  }
  fclose(file);
  freemem_addrows (&rmatind, &rmatval);
  free (row_name);
  return status;
}
```

```
/****** finds the number of senior IDs for Fix_Reverse_X1  *******/
// searches for id's that must not be assigned for training this month
int find_NoOfSeniorIDs(pilot_tag *pilot, session_tag *session)
{
   int num_seniors;
   pilot_tag *pltptr;
   session_tag *sesptr;

   num_seniors = 0;
   pltptr = PilotList;
   while (pltptr !=NULL) {
      if (pltptr->pilotindex == pilot->pilotindex) {
                        pltptr = pltptr->next_pilot;
                        continue;
            } // must be different id
            if (pltptr->position != pilot->position) {
                        pltptr = pltptr->next_pilot;
                        continue;
            }
            if (pltptr->expiration == 0){
                        pltptr = pltptr->next_pilot;
                        continue;
            } // must not expire in X
            if ((pltptr->expiration == 1) && (pltptr->seniority > pilot->seniority)){
                        pltptr = pltptr->next_pilot;
                        continue;
            } // must be more senior
      sesptr = pltptr->valid; // for each valid session of this id
            while (sesptr != NULL){
              if ((sesptr->index==session->index)&&(sesptr->priority==0))
              num_seniors++;
                     sesptr = sesptr->next_session;
            } // while (sesptr != NULL)
      pltptr = pltptr->next_pilot;
   } // while (pltptr != NULL)
   return(num_seniors);
}
```

```
/********finds the number of seniors from a different ID for X+1*********/
int NoOfdiffSeniorIDs(pilot_tag *pilot, session_tag *session)
{
   int num_seniors;
   pilot_tag *pltptr;
   session_tag *sesptr;

   num_seniors = 0;
   pltptr = PilotList;
   while (pltptr !=NULL) {
      if (pltptr->pilotindex == pilot->pilotindex) {
                        pltptr = pltptr->next_pilot;
                        continue;
            } // must be different id
            if (pltptr->position == pilot->position) {
                        pltptr = pltptr->next_pilot;
                        continue;
            }
            if (pltptr->expiration == 0){
                        pltptr = pltptr->next_pilot;
                        continue;
            } // must not expire in X
            if ((pltptr->expiration == 1) && (pltptr->seniority > pilot->seniority)){
                        pltptr = pltptr->next_pilot;
                        continue;
            } // must be more senior
      sesptr = pltptr->valid; // for each valid session of this id
            while (sesptr != NULL){
              if ((sesptr->index==session->index)&&(sesptr->priority==0))
              num_seniors++;
                    sesptr = sesptr->next_session;
            } // while (sesptr != NULL)
      pltptr = pltptr->next_pilot;
   } // while (pltptr != NULL)
   return(num_seniors);
}
```

```
/********   adds Reverse seniority constraints for X+1  ********/
int Fix_Reverse_X1()
{
   int pos, mm, status;
   int count, nzeros;
   pilot_tag *pltptr1, *pltptr2;
   session_tag *sesptr1, *sesptr2;

   FILE *file;
   file = fopen("file.txt","a+");
   sense = 'G';
   rhs = 0;
   status = 0;

   for (pos = 1; pos <=2 ; pos++){  // pos=1 for CP's and 2 for FO's
     pltptr1 = PilotList; // searches for id's expiring in X+1 that must be assigned for training in month X
           while (pltptr1 != NULL){
            if ((pltptr1->position != pos) || (pltptr1->expiration != 1)){
                     pltptr1 = pltptr1->next_pilot;
                     continue;
           }
      sesptr1 = pltptr1->valid; // for each valid session of this id
           while (sesptr1 != NULL){
             if (sesptr1->priority > 0) {
                           sesptr1 = sesptr1->next_session;
                           continue;
                  } // constraint added if this id has not bidded for this training session
                  count = find_NoOfSeniorIDs(pltptr1,sesptr1);
        nzeros = count;
                  fprintf(file,"count = %d\n", count);
                  if(count > 0 ){ // if count is positive a new constraint is added for this id
                    nzeros = nzeros + pltptr1->num_ofvalidbids; // add all the valid sessions for this id (left hand side)
                    getmem_addrows(&rmatind,&rmatval,nzeros);
```

```
                row_name = (char*) malloc(20*sizeof(char));
        mm = 0; // adds the constraint
                pltptr2 = PilotList;
                while(pltptr2 != NULL){
            if (pltptr1->pilotindex == pltptr2->pilotindex){
                                pltptr2 = pltptr2->next_pilot;
                                continue;
                        }
                        if (pltptr2->position !=pos) {
                                pltptr2 = pltptr2->next_pilot;
                                continue;
                        }
                    if (pltptr2->expiration == 0) {
                                pltptr2 = pltptr2->next_pilot;
                                continue;
                        }
                    if ((pltptr2->expiration == 1) && ((pltptr2->seniority) > (pltptr1->seniority))) {
                                pltptr2 = pltptr2->next_pilot;
                                continue;
                        }

                    sesptr2 = pltptr2->valid;
                    while (sesptr2 != NULL){
                      if ((sesptr1->index == sesptr2->index) && (sesptr2->priority == 0)){
                                rmatind[mm] = VarIndex[pltptr2->pilotindex-1][sesptr2->index-1];
                                rmatval[mm] = -1;
                                mm++;
                                break;
                              } //  end if loop
            sesptr2 = sesptr2->next_session;
                        } // end while (sesptr2 != NULL) loop
                    pltptr2 = pltptr2->next_pilot;
        }  // end while(pltptr2 != NULL)
```

```
                    sesptr2 = pltptr1->valid;
                    while(sesptr2 != NULL){
                      rmatind[mm] = VarIndex[pltptr1->pilotindex-1][sesptr2->index-1];
                            rmatval[mm] = count;
                            mm++;
          sesptr2 = sesptr2->next_session;
                    } // end while(sesptr2 != NULL)

                    sprintf(row_name, "FixRevID%dSES%d", pltptr1->pilotindex, sesptr1->index);
                    status = CPXaddrows(env, lp, 0, 1, nzeros, &rhs, &sense, &rmatbeg, rmatind, rmatval, NULL, &row_name);
                    if(status != 0){
                      printf("Failed to add Fix Reverse X1 constraint for pilot %d session %d, exiting\n", pltptr1->pilotindex, sesptr1->index);
                            return status;
                    }
                    freemem_addrows(&rmatind, &rmatval);
                    free(row_name);
                  }
                  sesptr1=sesptr1->next_session;
           }
           pltptr1=pltptr1->next_pilot;
          }
  }
  fclose(file);
  return status;
}

/********add's reverse seniority constraints for X+1 but for ID's that are different********/
int Fix_Reverse_diffID_X1()
{
  int pos, mm, status, TSET, nzcnt;
  int count, nzeros, indexofz;
  pilot_tag *pltptr1, *pltptr2;
  session_tag *sesptr1, *sesptr2, *sesptr3;
```

```
  double maxclasscrew;

  FILE *file;
  file = fopen("file.txt","a+");

  sense = 'G';
  rhs = 0;
  status = 0;

  getmem_newcols(&obj, &lb, &ub, &ctype, 1);
  obj[0] =0;
  lb[0]= 0;
  ub[0]=1;
  ctype[0]='B';

  for (pos = 1; pos <=2 ; pos++){  // pos=1 for CP's and 2 for FO's
    pltptr1 = PilotList; // searches for id's expiring in X+1 that must be assigned for training in month X
          while (pltptr1 != NULL){
           if ((pltptr1->position != pos) || (pltptr1->expiration != 1)){
                  pltptr1 = pltptr1->next_pilot;
                  continue;
           }

     sesptr1 = pltptr1->valid; // for each valid session of this id
          while (sesptr1 != NULL){
            if (sesptr1->priority > 0) {
                        sesptr1 = sesptr1->next_session;
                        continue;
                } // constraint added if this id has not bidded for this training session
            sesptr3 = SessionList;
                while (sesptr3 != NULL) {
                        if (sesptr3->index == sesptr1->index) {
                                if (pos == 1){
                                        //fprintf(file,"sescapacitycp=    %lf\n",    sesptr3->capacity_CP);
                                        maxclasscrew= sesptr3->capacity_CP;
```

```
                                }
                        if (pos ==2) {
                                //fprintf(file,"sescapacityfo=  %lf\n",  sesptr3-
>capacity_FO);
                                        maxclasscrew= sesptr3->capacity_FO;
                                }
                        }
                        sesptr3=sesptr3->next_session;
                }
                count = NoOfdiffSeniorIDs(pltptr1,sesptr1);
        nzeros = count;
                //fprintf(file,"count = %d\n", count);
                TSET=0;
                  pltptr2=PilotList;
                  while (pltptr2 != NULL) {
                        if ((pltptr2->position == pos) && (pltptr2->pilotindex != pltptr1-
>pilotindex)) {
                                if (pltptr2->expiration == 0) {
                                        sesptr2 = pltptr2->valid;
                                        while (sesptr2 != NULL) {
                                                if (sesptr2->index == sesptr1->index)
                                                        TSET++;
                                           sesptr2=sesptr2->next_session;
                                        }
                                }
                               if(pltptr2->expiration == 2) {
                                        sesptr2 = pltptr2->valid;
                                        while (sesptr2 != NULL) {
                                                if  ((sesptr2->index  ==  sesptr1->index)
&& (sesptr2->priority >0))
                                                        TSET++;
                                           sesptr2=sesptr2->next_session;
                                        }
                                }
                               if (pltptr2->expiration == 1) {
                                        sesptr2 = pltptr2->valid;
                                        while (sesptr2 != NULL) {
```

```
                                        if ((sesptr2->index == sesptr1->index)
&& ((sesptr2->priority >0) || (pltptr2->seniority>pltptr1->seniority)))
                                                TSET++;
                                    sesptr2=sesptr2->next_session;
                                }
                            }
                        }
                    pltptr2=pltptr2->next_pilot;
                }
            if (count > 0 )  { // if count is positive 1 new constraint is added for this
id
                    nzeros = nzeros + pltptr1->num_ofvalidbids; // add all the valid
sessions for this id (left hand side)
                    if (TSET>0)
                            getmem_addrows(&rmatind,&rmatval,nzeros+1);
                    else
                            getmem_addrows(&rmatind,&rmatval,nzeros);
                    row_name = (char*) malloc(64*sizeof(char));
        mm = 0; // adds the constraint
                pltptr2 = PilotList;
                while(pltptr2 != NULL){
                            if (pltptr1->pilotindex == pltptr2->pilotindex){
                                    pltptr2 = pltptr2->next_pilot;
                                continue;
                            }
                        if (pltptr2->position == pos) {
                                pltptr2 = pltptr2->next_pilot;
                                continue;
                            }
            if (pltptr2->expiration == 0) {
                                pltptr2 = pltptr2->next_pilot;
                                continue;
                            }
                    if ((pltptr2->expiration == 1) && ((pltptr2->seniority) > (pltptr1-
>seniority))) {
                                pltptr2 = pltptr2->next_pilot;
                                continue;
```

```
                    }
                    sesptr2 = pltptr2->valid;
                while (sesptr2 != NULL){
                        if ((sesptr1->index == sesptr2->index) &&
(sesptr2->priority == 0)){
                            rmatind[mm] = VarIndex[pltptr2-
>pilotindex-1][sesptr2->index-1];
                        rmatval[mm] = -1;
                        mm++;
                        break;
                          } //  end if loop
            sesptr2 = sesptr2->next_session;
                        } // end while (sesptr2 != NULL) loop
                pltptr2 = pltptr2->next_pilot;
                  }  // end while(pltptr2 != NULL)

                sesptr2 = pltptr1->valid;
                while(sesptr2 != NULL){
                        rmatind[mm]  =  VarIndex[pltptr1->pilotindex-1][sesptr2-
>index-1];
                    rmatval[mm] = count;
                    mm++;
            sesptr2 = sesptr2->next_session;
                  } // end while(sesptr2 != NULL)
                  if (TSET >0) {//add the variable z)
                        status=CPXnewcols(env ,lp, 1 ,obj,lb, ub, ctype, NULL);
                if (status != 0) {
                        printf("Failed to add the z variable,exiting\n");
                    return status;
                      }
                    rmatind[mm]=CPXgetnumcols(env,lp)-1;
                rmatval[mm]= count;
                indexofz=rmatind[mm];
                  }
        sprintf(row_name,  "FixRevdifX1ID%dSES%d",  pltptr1->pilotindex,  sesptr1-
>index);
                  if (TSET>0)
```

```
                        status = CPXaddrows(env, lp, 0, 1, nzeros+1, &rhs,
&sense, &rmatbeg, rmatind, rmatval, NULL, &row_name);
                else
                        status = CPXaddrows(env, lp, 0, 1, nzeros, &rhs,
&sense, &rmatbeg, rmatind, rmatval, NULL, &row_name);
                if(status != 0){
                   printf("Failed to add Fix Reverse X1 constraint for pilot %d
session %d, exiting\n", pltptr1->pilotindex, sesptr1->index);
                   return status;
                }
            freemem_addrows(&rmatind, &rmatval);
            free(row_name);
                if (TSET>0) {
                  TSET++;
                  getmem_addrows(&rmatind, &rmatval,TSET);
             row_name = (char*)malloc(64*sizeof(char));
             nzcnt=0;
             pltptr2=PilotList;
                  while (pltptr2 != NULL) {
                  if ((pltptr2->position == pos) && (pltptr2->pilotindex != pltptr1-
>pilotindex)) {
                        if (pltptr2->expiration == 0) {
                                sesptr2 = pltptr2->valid;
                                while (sesptr2 != NULL) {
                                        if (sesptr2->index == sesptr1->index) {
                                                rmatind[nzcnt]=VarIndex[pltptr2-
>pilotindex-1][sesptr2->index-1];
                                                rmatval[nzcnt]=1;
                                                nzcnt++;
                                        }
                                   sesptr2=sesptr2->next_session;
                                }
                        }
                      if(pltptr2->expiration == 2) {
                                sesptr2 = pltptr2->valid;
                                while (sesptr2 != NULL) {
```

```
                                        if ((sesptr2->index == sesptr1->index)
&& (sesptr2->priority > 0)) {
                                            rmatind[nzcnt]=VarIndex[pltptr2-
>pilotindex-1][sesptr2->index-1];
                                            rmatval[nzcnt]=1;
                                            nzcnt++;
                                        }
                                    sesptr2=sesptr2->next_session;
                                }
                            }
                        if (pltptr2->expiration == 1) {
                                sesptr2 = pltptr2->valid;
                            while (sesptr2 != NULL) {
                                        if ((sesptr2->index == sesptr1->index)
&& ((sesptr2->priority > 0) || (pltptr2->seniority>pltptr1->seniority))) {
                                            rmatind[nzcnt]=VarIndex[pltptr2-
>pilotindex-1][sesptr2->index-1];
                                            rmatval[nzcnt]=1;
                                            nzcnt++;
                                        }
                                    sesptr2=sesptr2->next_session;
                                }
                            }
                        }
                    pltptr2=pltptr2->next_pilot;
                        }

                        rmatind[nzcnt] = indexofz;
                        rmatval[nzcnt] = -maxclasscrew;

                        sprintf(row_name, "FixRevIzvarID%dSES%d", pltptr1-
>pilotindex, sesptr1->index);
                    status = CPXaddrows(env, lp, 0, 1, TSET, &rhs, &sense,
&rmatbeg, rmatind, rmatval, NULL, &row_name);
                    if(status != 0){
                      printf("Failed to add Fix Reverse X1 z constraint for pilot %d
session %d, exiting\n", pltptr1->pilotindex, sesptr1->index);
```

```
                    return status;
                }
                freemem_addrows(&rmatind, &rmatval);
                free(row_name);
                    }//IF TSET>0
            }//IF COUNT>0
               sesptr1 = sesptr1->next_session;
           } // end while (sesptr1 != NULL)
           pltptr1 = pltptr1->next_pilot;
  }// end while (pltptr1 != NULL)
 }// end for (pos = 1; pos <= 2; pos++)
  fclose(file);
  freemem_newcols(&obj,&lb,&ub,&ctype);
  return status;
}

/******  finds the number of senior IDs for Fix_Reverse_X2  ******/
// searches for id's that must not be assigned for training this month
int find_NoOfSeniorIDs2(pilot_tag *pilot, session_tag *session)
{
  int num_seniors;
  pilot_tag *pltptr;
  session_tag *sesptr;

  num_seniors = 0;
  pltptr = PilotList;
  while(pltptr != NULL){
        if (pltptr->pilotindex == pilot->pilotindex) {pltptr = pltptr->next_pilot;
continue;} // must be different id
        if (pltptr->position != pilot->position){pltptr = pltptr->next_pilot; continue;} //
must be same position
        if (pltptr->expiration != 2){pltptr = pltptr->next_pilot; continue;}  // must expire
in X+2
        if ((pltptr->seniority) > (pilot->seniority)){pltptr = pltptr->next_pilot; continue;}
// must be more senior
     sesptr = pltptr->valid; // for each vali8d session of this id
        while(sesptr != NULL){
```

```
                    if((sesptr->index == session->index) && (sesptr->priority == 0))
                            num_seniors++;
                    sesptr = sesptr->next_session;
            } // end while(sesptr != NULL)
        pltptr = pltptr->next_pilot;
    }// end while(pltptr != NULL)
    return(num_seniors);
}


/********   adds Reverse seniority constraints for X+2  ********/
int Fix_Reverse_X2()
{
    int pos, mm, status;
    int count, nzeros;
    pilot_tag *pltptr1, *pltptr2;
    session_tag *sesptr1, *sesptr2;

    sense = 'G';
    rhs = 0;
    status = 0;

    for(pos = 1; pos <=2 ; pos++){ // pos=1 for CP's and 2 for FO's
            pltptr1 = PilotList; // searches for id's expiring in X+2 that must be assigned
for training in month X
            while (pltptr1 != NULL){
                    if ((pltptr1->position != pos) || (pltptr1->expiration != 2)){pltptr1 =
pltptr1->next_pilot;  continue;}
                    sesptr1 = pltptr1->valid; // for each valid session of this id
                while (sesptr1 != NULL){
                            if (sesptr1->priority > 0) {sesptr1 = sesptr1->next_session;
continue;} // constraint added if this id has not bidded for this training session
                            count = find_NoOfSeniorIDs2(pltptr1,sesptr1);
            nzeros = count;
                            if(count > 0 ){ // if count is positive a new constraint is added
for this id
```

```
                                nzeros = nzeros + pltptr1->num_ofvalidbids;  // add all
the valid sessions for this id (left hand side)
                                getmem_addrows(&rmatind,&rmatval,nzeros);
                                row_name = (char*) malloc(23*sizeof(char));
                                mm = 0; // adds the constraint
                        pltptr2 = PilotList;
                        while(pltptr2 != NULL){
                                        if      (pltptr1->pilotindex      ==      pltptr2-
>pilotindex){pltptr2 = pltptr2->next_pilot; continue;}
                                if (pltptr2->position != pos) {pltptr2 = pltptr2->next_pilot;
continue;}
                                if (pltptr2->expiration != 2) {pltptr2 = pltptr2->next_pilot;
continue;}
                                if  ((pltptr2->seniority)  >  (pltptr1->seniority))  {pltptr2  =
pltptr2->next_pilot; continue;}
                                        sesptr2 = pltptr2->valid;
                                while (sesptr2 != NULL){
                                                if  ((sesptr1->index  ==  sesptr2->index)
&& (sesptr2->priority == 0)){
                                                        rmatind[mm]                     =
VarIndex[pltptr2->pilotindex-1][sesptr2->index-1];
                                                rmatval[mm] = -1;
                                                mm++;
                                                        break;
                                                } // end if loop
                        sesptr2 = sesptr2->next_session;
                                                } // end while (sesptr2 != NULL)
                                                pltptr2 = pltptr2->next_pilot;
                                        }  // end while(pltptr2 != NULL)
                                        sesptr2 = pltptr2->valid;
                        while(sesptr2 != NULL){
                                                rmatind[mm]   =   VarIndex[pltptr2->pilotindex-
1][sesptr2->index-1];
                                                rmatval[mm] = count;
                                                mm++;
                                sesptr2 = sesptr2->next_session;
                                        } // end while(sesptr2 != NULL)
```

```
                    sprintf(row_name, "FixRevIDII%dSES%d", pltptr1->pilotindex,
sesptr1->index);
                    status = CPXaddrows(env, lp, 0, 1, nzeros, &rhs, &sense,
&rmatbeg, rmatind, rmatval, NULL, &row_name);
                    if(status != 0){
                                printf("Failed to add Fix Reverse X2 constraint
for pilot %d session %d, exiting\n", pltptr1->pilotindex, sesptr1->index);
                        return status;
                          }
                    freemem_addrows(&rmatind, &rmatval);
            free(row_name);
                        } // end if(count > 0 ){
              sesptr1 = sesptr1->next_session;
                }  // end while (sesptr1 != NULL)
            pltptr1 = pltptr1->next_pilot;
          } // end while (pltptr1 != NULL)
  } // end for (pos = 1; pos <= 2; pos++)
  return status;
}

/********finds the number of seniors from a different ID for X+2*********/
int NoOfdiffSeniorIDsX2(pilot_tag *pilot, session_tag *session)
{
  int num_seniors;
  pilot_tag *pltptr;
  session_tag *sesptr;

  num_seniors = 0;
  pltptr = PilotList;
  while (pltptr !=NULL) {
    if (pltptr->pilotindex == pilot->pilotindex) {
                pltptr = pltptr->next_pilot;
                continue;
        } // must be different id
        if (pltptr->position == pilot->position) {
                pltptr = pltptr->next_pilot;
```

```
                continue;
            }
            if (pltptr->expiration != 2){
                pltptr = pltptr->next_pilot;
                continue;
            } // must not expire in X
            if  (pltptr->seniority > pilot->seniority){
                pltptr = pltptr->next_pilot;
                continue;
            } // must be more senior
      sesptr = pltptr->valid; // for each valid session of this id
            while (sesptr != NULL){
              if ((sesptr->index==session->index)&&(sesptr->priority==0))
              num_seniors++;
                  sesptr = sesptr->next_session;
            } // while (sesptr != NULL)
      pltptr = pltptr->next_pilot;
   } // while (pltptr != NULL)
   return(num_seniors);
}


/********add's reverse seniority constraints for X+2 but for ID's that are
different********/
int Fix_Reverse_diffID_X2()
{
   int pos, mm, status, TSET, nzcnt;
   int count, nzeros, indexofz;
   pilot_tag *pltptr1, *pltptr2;
   session_tag *sesptr1, *sesptr2, *sesptr3;
   double maxclasscrew;


   FILE *file;
   file = fopen("file.txt","a+");
```

```
  sense = 'G';
  rhs = 0;
  status = 0;

  getmem_newcols(&obj, &lb, &ub, &ctype, 1);
  obj[0] =0;
  lb[0]= 0;
  ub[0]=1;
  ctype[0]='B';

  for (pos = 1; pos <=2 ; pos++){  // pos=1 for CP's and 2 for FO's
    pltptr1 = PilotList; // searches for id's expiring in X+1 that must be assigned for
training in month X
          while (pltptr1 != NULL){
           if ((pltptr1->position != pos) || (pltptr1->expiration != 2)){
                    pltptr1 = pltptr1->next_pilot;
                    continue;
          }

     sesptr1 = pltptr1->valid; // for each valid session of this id
          while (sesptr1 != NULL){
            if (sesptr1->priority > 0) {
                        sesptr1 = sesptr1->next_session;
                        continue;
                } // constraint added if this id has not bidded for this training session
            sesptr3 = SessionList;
                while (sesptr3 != NULL) {
                        if (sesptr3->index == sesptr1->index) {
                                if (pos == 1){
                                        maxclasscrew= sesptr3->capacity_CP;
                                }
                      if (pos ==2) {
                                        maxclasscrew= sesptr3->capacity_FO;
                                }
                        }
                        sesptr3=sesptr3->next_session;
                }
```

```
                count = NoOfdiffSeniorIDsX2(pltptr1,sesptr1);
        nzeros = count;
                TSET=0;
                  pltptr2=PilotList;
                  while (pltptr2 != NULL) {
                          if ((pltptr2->position == pos) && (pltptr2->pilotindex != pltptr1-
>pilotindex)) {
                                  if (pltptr2->expiration <= 1) {
                                          sesptr2 = pltptr2->valid;
                                          while (sesptr2 != NULL) {
                                                  if (sesptr2->index == sesptr1->index)
                                                          TSET++;
                                             sesptr2=sesptr2->next_session;
                                          }
                                  }

                                  if (pltptr2->expiration == 2) {
                                          sesptr2 = pltptr2->valid;
                                          while (sesptr2 != NULL) {
                                                  if ((sesptr2->index == sesptr1->index)
&& ((sesptr2->priority >0) || (pltptr2->seniority>pltptr1->seniority)))
                                                          TSET++;
                                             sesptr2=sesptr2->next_session;
                                          }
                                  }
                           }
                     pltptr2=pltptr2->next_pilot;
                  }
                if (count > 0 ) { // if count is positive a new constraint is added for this
id
                        nzeros = nzeros + pltptr1->num_ofvalidbids; // add all the valid
sessions for this id (left hand side)
                        if (TSET >0)
                                getmem_addrows(&rmatind,&rmatval,nzeros+1);
                        else
                                getmem_addrows(&rmatind,&rmatval,nzeros);
                   row_name = (char*) malloc(64*sizeof(char));
```

```
mm = 0; // adds the constraint
        pltptr2 = PilotList;
        while(pltptr2 != NULL){
                if (pltptr1->pilotindex == pltptr2->pilotindex){
                        pltptr2 = pltptr2->next_pilot;
                   continue;
                }
              if (pltptr2->position == pos) {
                        pltptr2 = pltptr2->next_pilot;
                   continue;
                }
      if (pltptr2->expiration != 2) {
                        pltptr2 = pltptr2->next_pilot;
                   continue;
                }
           if ((pltptr2->seniority) > (pltptr1->seniority)) {
                        pltptr2 = pltptr2->next_pilot;
                   continue;
                }

           sesptr2 = pltptr2->valid;
           while (sesptr2 != NULL){
                        if ((sesptr1->index == sesptr2->index) &&
(sesptr2->priority == 0)){
                                rmatind[mm] = VarIndex[pltptr2-
>pilotindex-1][sesptr2->index-1];
                rmatval[mm] = -1;
                     mm++;
                     break;
                     } //  end if loop
      sesptr2 = sesptr2->next_session;
                } // end while (sesptr2 != NULL) loop
           pltptr2 = pltptr2->next_pilot;
             }  // end while(pltptr2 != NULL)

        sesptr2 = pltptr1->valid;
        while(sesptr2 != NULL){
```

```
                    rmatind[mm] = VarIndex[pltptr1->pilotindex-1][sesptr2->index-1];
                    rmatval[mm] = count;
                    mm++;
            sesptr2 = sesptr2->next_session;
                } // end while(sesptr2 != NULL)
            if (TSET>0) {
                    status=CPXnewcols(env ,lp, 1 ,obj,lb, ub, ctype, NULL);
              if (status != 0) {
                    printf("Failed to add the z variable,exiting\n");
                    return status;
                    }
            rmatind[mm]=CPXgetnumcols(env,lp)-1;
              rmatval[mm]= count;
              indexofz=rmatind[mm];
                }
            sprintf(row_name, "FixRevdifX2ID%dSES%d", pltptr1->pilotindex, sesptr1->index);
                if (TSET>0)
                    status = CPXaddrows(env, lp, 0, 1, nzeros+1, &rhs, &sense, &rmatbeg, rmatind, rmatval, NULL, &row_name);
          else
                    status = CPXaddrows(env, lp, 0, 1, nzeros, &rhs, &sense, &rmatbeg, rmatind, rmatval, NULL, &row_name);
            if(status != 0){
              printf("Failed to add Fix Reverse X2 constraint for pilot %d session %d, exiting\n", pltptr1->pilotindex, sesptr1->index);
                    return status;
                }
            freemem_addrows(&rmatind, &rmatval);
            free(row_name);
            if (TSET>0 ) {
              TSET++;
                  getmem_addrows(&rmatind, &rmatval,TSET);
              row_name = (char*)malloc(64*sizeof(char));
              nzcnt=0;
              pltptr2=PilotList;
```

```
                        while (pltptr2 != NULL) {
                        if ((pltptr2->position == pos) && (pltptr2->pilotindex != pltptr1-
>pilotindex)) {
                                if (pltptr2->expiration <= 1) {
                                        sesptr2 = pltptr2->valid;
                                        while (sesptr2 != NULL) {
                                                if (sesptr2->index == sesptr1->index) {
                                                        rmatind[nzcnt]=VarIndex[pltptr2-
>pilotindex-1][sesptr2->index-1];
                                                   rmatval[nzcnt]=1;
                                                   nzcnt++;
                                                }
                                           sesptr2=sesptr2->next_session;
                                        }
                                }

                           if (pltptr2->expiration == 2) {
                                        sesptr2 = pltptr2->valid;
                                   while (sesptr2 != NULL) {
                                                if ((sesptr2->index == sesptr1->index)
&& ((sesptr2->priority > 0) || (pltptr2->seniority>pltptr1->seniority))) {
                                                        rmatind[nzcnt]=VarIndex[pltptr2-
>pilotindex-1][sesptr2->index-1];
                                                   rmatval[nzcnt]=1;
                                                   nzcnt++;
                                                }
                                           sesptr2=sesptr2->next_session;
                                        }
                                }
                        }
                     pltptr2=pltptr2->next_pilot;
                        }

                        rmatind[nzcnt] = indexofz;
                        rmatval[nzcnt] = -maxclasscrew;
```

```
                        sprintf(row_name, "FixRevIIzvarID%dSES%d", pltptr1->pilotindex, sesptr1->index);
                        status = CPXaddrows(env, lp, 0, 1, TSET, &rhs, &sense, &rmatbeg, rmatind, rmatval, NULL, &row_name);
                        if(status != 0){
                          printf("Failed to add Fix Reverse X1 z constraint for pilot %d session %d, exiting\n", pltptr1->pilotindex, sesptr1->index);
                            return status;
                        }

                        freemem_addrows(&rmatind, &rmatval);
                        free(row_name);
                            }//IF TSET>0
                      }//IF COUNT>0
              sesptr1 = sesptr1->next_session;
            } // end while (sesptr1 != NULL)
            pltptr1 = pltptr1->next_pilot;
   }// end while (pltptr1 != NULL)
  }// end for (pos = 1; pos <= 2; pos++)
   fclose(file);
   freemem_newcols(&obj,&lb,&ub,&ctype);
   return status;
}
/*******   adds no empty seats constraint  ********/
int no_empty_seats()
{
   int  mm, status;
   pilot_tag  *pltptr;

   if (Not_Expire == 0)
      return 0;

   getmem_newcols(&obj, &lb, &ub, &ctype,1);
   obj[0] = 0;
   lb[0] = 0;
   ub[0] = Not_Expire;
   ctype[0] ='I';
```

```
  status = CPXnewcols(env, lp, 1, obj, lb, ub, ctype, NULL);
  if (status != 0){
    printf("Failed to add no empty seats var, exiting\n");
          return status;
  }
  freemem_newcols(&obj, &lb, &ub, &ctype);
  getmem_addrows(&rmatind, &rmatval, Not_Expire+1);
  sense = 'E';
  rhs = 0;
  mm = 0; // searches for the vars representing the non-assignment
  pltptr = PilotList;
  while (pltptr != NULL) {
    if(pltptr->expiration > 0){
          rmatind[mm] = tot_valid_bids + mm;
          rmatval[mm] = -1;
          mm++;
        }
        pltptr = pltptr->next_pilot;
  } // end while (pltptr != NULL)
  rmatind[mm] = CPXgetnumcols(env, lp)-1; // index of the penalty variable
  rmatval[mm]= 1; // the penalty variable is set equal to the number of unassigned ids
  AddToPenaltyList(CPXgetnumcols(env, lp)-1, -1,-1, -1, 'M', &PenaltyList);
  row_name = (char*) malloc(10*sizeof(char));
  sprintf(row_name, "NOEMS");
  status = CPXaddrows(env, lp, 0, 1, Not_Expire + 1, &rhs, &sense, &rmatbeg,
rmatind, rmatval, NULL, &row_name);
  if (status != 0){
    printf("Failed to add no empty seat constraint for pilot%d, exiting\n", pltptr-
>pilotindex);
          return status;
  }

  freemem_addrows(&rmatind, &rmatval);
  free(row_name);
  return status;
}
```

```
/************finds the number of variables in each balance constraint**********/
/*int find_number_of_variables (int jjj, int kkk, int poss)
{
        int num_variables;
        pilot_tag *pltptr;
        session_tag *sesptr;

        num_variables= 0;
        pltptr = PilotList;
   while(pltptr != NULL) {
                if ((pltptr->position == poss) || (poss==0)) {
                        sesptr= pltptr->valid;
                        while (sesptr !=NULL) {
                                if (sesptr->index == jjj)
                                        num_variables++;
                                if (sesptr->index == kkk)
                                        num_variables ++;
                                sesptr=sesptr->next_session;
                        }
                }
                pltptr=pltptr->next_pilot;
        }
        return(num_variables);
}
*/
/***********adds balance constraints**********/
int balance(int balancelevel)
{
        int pos, jj, status;
        int nzcnt ,count;
        pilot_tag *pltptr;
        session_tag *sesptr;
        session_tag *sesptr1;

        status = 0;
```

```
        rhs = 0;
        sense = 'E';

        FILE *file1;
    file1 = fopen("file1.txt","a+");

        row_name = (char*)malloc(64*sizeof(char));
        getmem_addrows(&rmatind, &rmatval, (2*N)+2);
        getmem_newcols(&obj, &lb, &ub, &ctype,2);
        obj[0]=0.05;
lb[0]=0;
        ub[0]=CPX_INFBOUND;
        ctype[0]= 'I';
        obj[1] = 0.05;
        lb[1] = 0;
        ub[1] = CPX_INFBOUND;
        ctype[1] = 'I';
        if (balancelevel == 2) {//heavy balance
                for (pos = 0; pos<=3; pos++){
                        sesptr=SessionList;
                        while (sesptr != NULL) {
                                nzcnt = 0;
                                pltptr = PilotList;
                                while (pltptr != NULL) {
                                        if ((pltptr->position != pos) && (pos >0)) {
                                                pltptr=pltptr->next_pilot;
                                                continue;
                                        }
                                        sesptr1=pltptr->valid;
                                        while (sesptr1 != NULL) {
                                                if (sesptr1->index == sesptr->index) {
                                                        rmatind[nzcnt]  =  VarIndex[pltptr-
>pilotindex-1][sesptr1->index-1];
                                                        rmatval[nzcnt]=1;
                                                        nzcnt ++;
                                                        break;
                                                }//end if loop;
```

```
                                        sesptr1= sesptr1->next_session;
                                    }//end while sesptr1!=NULL
                pltptr= pltptr->next_pilot;
                                    }//end while pltptr !=NULL
                                    if (nzcnt == 0) {
                                        sesptr= sesptr->next_session;
                                        continue;
                                    }
                                    count= nzcnt;
                                    for (jj =sesptr->index+1; jj <= S; jj++) {
                                        nzcnt = count;
                                   pltptr = PilotList;
                                   while (pltptr != NULL) {
                                            if ((pltptr->position != pos) && (pos >0)) {
                                                pltptr=pltptr->next_pilot;
                                               continue;
                                            }
                                          sesptr1=pltptr->valid;
                                          while (sesptr1 != NULL) {
                                                    if (sesptr1->index == jj) {
                                                            rmatind[nzcnt]              =
VarIndex[pltptr->pilotindex-1][sesptr1->index-1];
                                                       rmatval[nzcnt]= - 1;
                                                       nzcnt ++;
                                                       break;
                                                    }//end if loop;
                                               sesptr1= sesptr1->next_session;
                                            }//end while sesptr1!=NULL
                  pltptr= pltptr->next_pilot;
                                    }//end while pltptr !=NULL
                                    if (nzcnt== count) continue;

                                    status = CPXnewcols(env, lp, 2, obj, lb, ub,
ctype, NULL);
                                    if (status != 0) {
                                        printf ("Failed to add variable balance
v,exiting\n");
```

```
                                        return status;
                                    }

                                    rmatind[nzcnt]= CPXgetnumcols(env,lp)-2;
                                    rmatval[nzcnt] = 1;
                                    nzcnt++;
                                    rmatind[nzcnt]= CPXgetnumcols(env,lp)-1;
                                    rmatval[nzcnt] =-1;
                                    nzcnt++;

                                    sprintf(row_name,
"balance_constraintPOS%dSES%dSES%d",pos, sesptr->index ,jj);
                                    status=CPXaddrows(env,lp,0,1,nzcnt,&rhs,
&sense,&rmatbeg,rmatind,rmatval,NULL,&row_name);
                                    if (status != 0) {
                                        printf ("Failed to add balance constraint
v,exiting\n");
                                        return status;
                                    }
                                }//end for jj
                                sesptr=sesptr->next_session;
                            }//end while sesptr != NULL
                        }//end for pos
                }
            if (balancelevel == 1) {//light balance
                        for (pos = 0; pos<=2; pos++) {
                            sesptr= SessionList;
                            while(sesptr != NULL) {
                                nzcnt=0;
                                pltptr=PilotList;
                                while(pltptr!=NULL) {
                                    if ((pos == 1) && (pltptr->position >2)) {
                                        pltptr=pltptr->next_pilot;
                                        continue;
                                    }
                                    if ((pos ==2 ) && (pltptr->position <3)) {
                                        pltptr=pltptr->next_pilot;
```

```
                    continue;
                }
                sesptr1=pltptr->valid;
                while (sesptr1 !=NULL ) {
                    if (sesptr1->index == sesptr->index) {
                        rmatind[nzcnt]  =  VarIndex[pltptr-
>pilotindex-1][sesptr1->index-1];
                        rmatval[nzcnt] = 1;
                        nzcnt++;
                        break;
                    }
                    sesptr1=sesptr1->next_session;
                }//end while (sesptr1 !=NULL )
                pltptr=pltptr->next_pilot;
            }//end while (pltptr !=NULL )
            if (nzcnt == 0) {
                sesptr=sesptr->next_session;
                continue;
            }
            count=nzcnt;
            for (jj= sesptr->index+1; jj<=S;jj++) {
                nzcnt = count;
                pltptr=PilotList;
              while(pltptr!=NULL) {
                    if ((pos == 1) && (pltptr->position >2)) {
                        pltptr=pltptr->next_pilot;
                      continue;
                    }
                  if ((pos ==2 ) && (pltptr->position <3)) {
                      pltptr=pltptr->next_pilot;
                      continue;
                    }
                  sesptr1=pltptr->valid;
                  while (sesptr1 !=NULL ) {
                      if (sesptr1->index == jj) {
                                rmatind[nzcnt]              =
VarIndex[pltptr->pilotindex-1][sesptr1->index-1];
```

```
                                        rmatval[nzcnt] = -1;
                                        nzcnt++;
                                        break;
                                }
                                sesptr1=sesptr1->next_session;
                        }//end while (sesptr1 !=NULL )
                        pltptr=pltptr->next_pilot;
                }//end while (pltptr !=NULL )
                if (nzcnt == count) continue;

                status = CPXnewcols(env, lp, 2, obj, lb, ub, ctype,
NULL);
                        if (status != 0) {
                                printf ("Failed to add variable balance
v,exiting\n");
                                return status;
                        }

                        rmatind[nzcnt]= CPXgetnumcols(env,lp)-2;
                        rmatval[nzcnt] = 1;
                        nzcnt++;
                        rmatind[nzcnt]= CPXgetnumcols(env,lp)-1;
                        rmatval[nzcnt] =-1;
                        nzcnt++;

                        sprintf(row_name,
"balance_constraintPOS%dSES%dSES%d",pos, sesptr->index ,jj);
                        status=CPXaddrows(env,lp,0,1,nzcnt,&rhs,
&sense,&rmatbeg,rmatind,rmatval,NULL,&row_name);
                        if (status != 0) {
                                printf ("Failed to add balance constraint
v,exiting\n");
                                return status;
                        }
                }//end for jj
                sesptr=sesptr->next_session;
        }//end while sesptr != NULL
```

```
                }//end for pos
        }//end if balancelevel=1
        freemem_newcols(&obj, &lb, &ub, &ctype);
        freemem_addrows (&rmatind,&rmatval);
        free(row_name);
        fclose(file1);
        return status;
}

/********** adds language constraints ********/
// ensures that each id's assignment will be compatible with this session language
// also finds a feasible language assignment to each session which is free

int add_language()
{
  int lang, nzcnt;
  int status;
  int flag;
  pilot_tag *pltptr;
  session_tag *sesptr, *sesptr2;

  getmem_addrows(&rmatind, &rmatval, N+1);
  getmem_newcols(&obj, &lb, &ub, &ctype, 1); // allocate memory for languge defining variable

  obj[0] = 0;
  lb[0] = 0;
  ub[0] = 1;
  ctype[0] ='B'; // Binary variable, uj
  sense = 'L' ;
  sesptr = SessionList; // for each training session
  while (sesptr != NULL){
    flag=0; // flag becomes 1 when language defining variable has been set for the current session
    for (lang = 1; lang <=2 ; lang++){ // lang=1 is for FR and 2 for EN
            nzcnt = 0;     // counts the id's which speak this language (lang) and are valid for this session
```

```
            pltptr = PilotList;
            while (pltptr != NULL){
               if(pltptr->language == lang){
                  sesptr2 = pltptr -> valid;
                        while(sesptr2 != NULL){
                           if(sesptr->index == sesptr2->index){
                                    rmatind[nzcnt] = VarIndex[pltptr->pilotindex-1][sesptr-
>index-1];
                                    rmatval[nzcnt] = 1;
                                    nzcnt++;
                                          }
               sesptr2 = sesptr2->next_session;
                                    } //end while(sesptr2!=NULL) loop
                           } // end if(pltptr->language == lang) loop
         pltptr = pltptr-> next_pilot;
                  } // end while (pltptr != NULL)
                  if (nzcnt > 0){
               if (flag == 0){
            status = CPXnewcols(env, lp, 1, obj, lb, ub, ctype, NULL);
            if (status != 0){
                     printf("Failed to add language var, exiting\n");
                     return status;
                                    }
            else flag =1; // flag becomes 1 because language defining variable has
been set for the current session
                           } // end if (flag == 0) loop
                        rmatind[nzcnt] = CPXgetnumcols(env, lp)-1; // column index of
language defining variable
               if(lang == 1){ // for French speaking
                  rmatval[nzcnt] = -nzcnt;
                        rhs = 0;
                        row_name = (char*) malloc(20*sizeof(char));
                        sprintf(row_name, "LANG1ses%d",sesptr->index);
                          }
         else{ // for English speaking
                     rmatval[nzcnt] = nzcnt;
                        rhs = nzcnt;
```

```
                        row_name = (char*) malloc(20*sizeof(char));
                        sprintf(row_name, "LANG2ses%d",sesptr->index);
                          }
                      status = CPXaddrows(env, lp, 0, 1, nzcnt + 1, &rhs, &sense,
&rmatbeg, rmatind, rmatval, NULL, &row_name);
                        if (status != 0){
              printf("Failed to add language for session%d, exiting\n", sesptr->index);
                    return status;
                        }
                    }// end if (nzcnt > 0) loop
          }//end for lang loop
          sesptr=sesptr->next_session;
  } //end while (sesptr != NULL) loop
  freemem_newcols(&obj, &lb, &ub, &ctype);
  freemem_addrows(&rmatind, &rmatval);
  free(row_name);
  return status;
}

/************  evaluates and solves the problem  ***********/
int solve_evaluate_results()
{
  int index, status;
  int solstat, colnum;
  double objval;
  double *sol;
  pilot_tag *pltptr;
  session_tag *sesptr;

  FILE *file;
  file = fopen("file.txt","a+"); /* apend file (add text to
  a file or create a file if it does not exist.*/

  status = CPXmipopt(env,lp); // solves the problem
  if (status != 0){
    printf ("Failed to  to find a solution to the problem %d.\n", status);
          return status;
```

```
  }

  solstat = CPXgetstat(env,lp); // accesses the solution status of the problem
  if (solstat == 103){
          printf(" Problem is not feasible, exiting\n");
     status = 0;
          return status;
  }

  status = CPXgetobjval(env ,lp, &objval); // accesses the solution of objective value
  if (status != 0){
          printf("Failed to access the solution objective value, exiting\n");
          return status;
  }

  colnum = CPXgetnumcols(env, lp); // memory allocation of solution array
  fprintf(file,"number of variables %d\n",colnum);
  sol = (double *)malloc(colnum * sizeof(double));

  status = CPXgetx(env, lp, sol, 0, colnum-1); // accesses the solution values for a
range of problem variables
  if (status != 0){
    printf("Failed to access the solution, exiting\n");
          return status;
  }

  index = 0;
  pltptr = PilotList;
  while (pltptr != NULL) {
    sesptr = pltptr->valid;
          while(sesptr != NULL){
                  if(sol[index] > 0.5)
                          fprintf(file,"Pilot %d assigned to session %d\n", pltptr-
>pilotindex, sesptr->index);
                  index++;
                  sesptr = sesptr->next_session;
          } // end while(sesptr != NULL)
```

```
            pltptr = pltptr->next_pilot;
    } // end while (pltptr != NULL)

    pltptr = PilotList;
    while (pltptr != NULL) {
            if (pltptr -> expiration > 0){
                    if (sol[index] > 0.5)
                            fprintf(file,"Pilot %d remains unassigned\n", pltptr-
>pilotindex);
                    index++;
            }
            pltptr = pltptr->next_pilot;
    } // end while (pltptr != NULL)

    fprintf(file,"%d pilots remain non-assigned\n", (int)sol[index]);
    fprintf(file,"%d seats remain empty\n", (int)total_seats-(N-(int)sol[index]));
    index++;

    sesptr = SessionList;
    while(sesptr != NULL){
      if(sol[index] > 0.5)
        fprintf(file,"The language of session %d is french\n", sesptr->index);
          else
                  fprintf(file,"The language of session %d is english\n", sesptr->index);
          index++;
          sesptr = sesptr->next_session;
    } // end while(sesptr != NULL)
    while (index <= CPXgetnumcols(env,lp)-1) {
            fprintf(file, "variable= %lf\n", sol[index]);
            index++;
    }

    fprintf(file,"Optimal objective = %Lf\n", objval);
    free(sol);
    fclose(file);
    return(status);
}
```

```
/****** releases the problem ******/
int free_problem()
{
   int status;
   pilot_tag  *pltptr1, *pltptr2;
   session_tag *ses1, *ses2;

   pltptr1 = PilotList;
   while (pltptr1 != NULL){
      ses1 = pltptr1->valid;
           while (ses1 != NULL){
           ses2 = ses1->next_session;
            free(ses1);
           ses1 = ses2;
           }
           pltptr2 = pltptr1->next_pilot;
           free(pltptr1);
           pltptr1 = pltptr2;
} // end while (pltptr1 != NULL)

status = CPXfreeprob (env, &lp);
if (status != 0)
printf ("CPXfreeprob failed, error code %d.\n", status);

status = CPXcloseCPLEX (&env);
if (status != 0)
printf ("CPXclose env, error code %d.\n", status);

return status;
}

 /***********  Do  Penalty **********/
// assigns penalties to the PenaltyList nodes with indices from k to last and returns the index
// k is set equal to last-Num Penalties and is decreased until it refers to the first node of the id
```

```
Coeff_tag *DoPenalty(Coeff_tag  *last)
{
  int  index, status, solstat, numpenalties;
  double  Diff, Penalty, Increment, SSum;
  Coeff_tag  *prev, *next, *first, *newlast;
  double  *sol, *penaltyarray;

  FILE *file;
  file = fopen("file.txt","a+"); /* apend file (add text to
  a file or create a file if it does not exist.*/
  penaltyarray = (double *)malloc(CPXgetnumcols(env,lp) * sizeof(double));

  numpenalties = 100; // aproxiamate number of penalty nodes that will be assigned
  index = 1;
  first = last;
  prev = first->LeftLink;

  while ((prev != NULL) && (index < numpenalties)){
    first = prev;
    prev = prev->LeftLink;
    index++;
        //printf("Index=%d\n",index);
  } // end while loop

  prev = first->LeftLink;
  while ((prev != NULL) && (prev->Sen == first->Sen)){
    first = prev;
    prev = prev->LeftLink;
  }
  //fprintf(file,"first->Sen=%d\n",first->Sen);
  //fprintf(file,"first->ILogIdx=%d\n",first->ILogIdx);
  newlast = prev;
  //fprintf(file,"newlast->Sen=%d\n",newlast->Sen);
  //fprintf(file,"newlast->ILogIdx=%d\n",newlast->ILogIdx);
  Increment = 1;
  Penalty = -Increment;
  Penalty = 0;
```

```
SSum = 0; // stores the sum of the highest penalties of the id's
Diff = Increment; // difference between 2 consecutive penalties of the same id

if (first == last){
  Penalty = 0;
  status = CPXchgcoef(env, lp, -1, first->ILogIdx, Penalty);
       penaltyarray[first->ILogIdx]=Penalty;
  if (status != 0){
    printf ("Failed to change penalty.\n");
    newlast = NULL;
              return newlast;
       }
} // end if(first == last)
else {
  prev = first->LeftLink;
  next = first;
  while (prev != last){
    if ((prev == NULL) || (prev->Sen != next->Sen)){//if differrent id
                    fprintf(file,"Case Differrent Id\n");
                fprintf(file,"next->ILogIdx=%d\n",next->ILogIdx);
                fprintf(file,"next->Sen=%d\n",next->Sen);
      //fprintf(file,"SSum=%5.0Lf\n",SSum);
      //fprintf(file,"Penalty=%5.0Lf\n",Penalty);
                    SSum = SSum + Penalty;
                Diff = SSum + Increment;
                Penalty = 0;
                    fprintf(file,"SSum=SSum + Penalty=%5.0Lf\n",SSum);
                fprintf(file,"Diff=SSum + Increment= %5.0Lf\n\n",Diff);
                //getchar();


                    if (next->RecType == 'M')
                        status = CPXchgcoef(env, lp, -1, next->ILogIdx, Diff);
                    else
          status = CPXchgcoef(env, lp, -1, next->ILogIdx, Penalty);
                    if (status != 0){
                        printf ("Failed to change penalty.\n");
```

```
                        newlast = NULL;
                    return newlast;
                        }
                } // end if loop
                else{// if same id
                        fprintf(file,"Case Same Id\n");
                   fprintf(file,"next->ILogIdx=%d\n",next->ILogIdx);
                   fprintf(file,"next->Sen=%d\n",next->Sen);

                        //fprintf(file,"Diff=%5.0Lf\n\n",Diff);
                   //fprintf(file,"Penalty=%5.0Lf\n\n",Penalty);
                        if ((next == first) || (next->Prior < prev->Prior))
                                Penalty = Penalty + Diff;
                        fprintf(file,"Penalty = Penalty + Diff=%5.0Lf\n\n",Penalty);
                   //getchar();
                        status = CPXchgcoef(env, lp, -1, next->ILogIdx, Penalty);

                        if (status != 0){
                                printf ("Failed to change penalty.\n");
                                newlast = NULL;
                    return newlast;
                        }
                } // end else loop
                penaltyarray[next->ILogIdx]=Penalty;
     prev = next;
     next = next->RightLink;
        } // end while (prev != last)
} // end else loop

CPXwriteprob(env, lp, "debug.lp", NULL);
status = CPXmipopt(env,lp); // solves the problem
if (status != 0){
  printf ("Mipopt failed.\n");
  newlast = NULL;
  return newlast;
}
solstat = CPXgetstat(env,lp); // accesses the solution status of the problem
```

```
  if (solstat == 103){
    printf ("Problem infeasible.\n");
          newlast = NULL;
                    return newlast;
                      }
  sol = (double *)malloc(CPXgetnumcols(env,lp) * sizeof(double));
  status = CPXgetx(env, lp, sol, 0, CPXgetnumcols(env,lp)-1); // accesses the solution values for a range of problem variables
  if(status != 0){
    printf("Failed to access the solution objective value, exiting\n");
        newlast = NULL;
                    return newlast;
                      }
  getmem_addrows(&rmatind, &rmatval, 1);
  sense = 'E';  rmatval[0] = 1;

  objective=objective+penaltyarray[first->ILogIdx]*sol[first->ILogIdx];
  status = CPXchgcoef(env, lp, -1, first->ILogIdx, 0); // changes a single coefficient in the constraint matrix
  if (status != 0){
    printf ("Failed to change penalty.\n");
          newlast = NULL;
                    return newlast;
                      }
  rmatind[0] = first->ILogIdx;
  rhs = sol[first->ILogIdx];

  if (first->RecType == 'M' || first->Prior > 0){
    status = CPXaddrows(env,lp,0,1,1,&rhs,&sense,&rmatbeg,rmatind,rmatval,NULL,NULL);
          if (status != 0){
                printf ("Failed to change penalty.\n");
                newlast = NULL;
                      return newlast;
                        }
  }
```

```
  prev = first;
  next = prev->RightLink;
  while (prev != last){
          objective=objective+penaltyarray[next->ILogIdx]*sol[next->ILogIdx];
              fprintf(file,"penalty %lf\n",penaltyarray[next->ILogIdx]);
              fprintf(file,"sol %lf\n",sol[next->ILogIdx]);
      status = CPXchgcoef(env, lp, -1, next->ILogIdx, 0);
      if (status != 0){
                        printf ("Failed to change penalty.\n");
                        newlast = NULL;
                      return newlast;
                        }
          rmatind[0] = next->ILogIdx;
          rhs = sol[next->ILogIdx];
          if (next->RecType == 'M' || next->Prior > 0){
                        status=
CPXaddrows(env,lp,0,1,1,&rhs,&sense,&rmatbeg,rmatind,rmatval,NULL,NULL);
                    if (status != 0){
                                  printf ("Failed to change penalty.\n");
                                  newlast = NULL;
                      return newlast;
                        }
                  }
      prev = next;
      next = next->RightLink;
  }// end while (prev != last)
  fprintf(file,"objective= %lf\n",objective);
  freemem_addrows(&rmatind,&rmatval);
  free(sol);
  free(penaltyarray);
  fclose(file);
  CPXwriteprob(env, lp, "bid4.lp", NULL);
  printf("DoPenaltyExit\n");
  //getchar();
  return newlast;
}
/************ main program ***********/
```

```
int main (int argc, char **argv)
{
   int  status, start, stop, Balancelevel;
   long double duration;

   FILE *file1;
   file1 = fopen("file1.txt","w+");

   status = init_problem();
   if (status != 0){
            printf("init_problem failed, exiting\n");
return status;
}

   status = fill_lists();
   if (status != 0){
            printf("fill_lists failed, exiting\n");
return status;
   }

   status = print_lists();
   if (status != 0){
            printf("print_lists failed, exiting\n");
          return status;
   }

   start = clock();

   status = add_columns();
   if (status != 0){
            printf("Add_columns failed, exiting\n");
          return status;
}

  /* status = CPXwriteprob(env, lp, "1979.lp", NULL);
   if (status != 0){
            printf ("Failed to write problem %d.\n", status);
```

```
        return status;
  }*/


  status = add_assignments();
  if (status != 0){
          printf("Add_assignments failed, exiting\n");
return status;
  }

  status = add_capacity_trainees();
  if (status != 0){
     printf("Add_total capacity failed, exiting\n");
return status;
  }

  status = add_capacity_cockpit_cabin();
   if (status != 0){
     printf("Add_capacity in cockpit and in cabin failed, exiting\n");
return status;
  }

   status = add_capacity_per_position();
if (status != 0){
     printf("Add_capacity per position failed, exiting\n");
return status;
  }

        status = no_empty_seats();
        if (status != 0){
          printf("No empty seats failed, exiting\n");
     return status;
  }

        status = add_language();
  if (status != 0){
          printf("Add language failed, exiting\n");
```

```
return status;
  }

  status = Fix_Reverse_X1();
  if (status != 0){
     printf("Fix_Reverse_X1 failed, exiting\n");
        return status;
  }

  status = Fix_Reverse_diffID_X1();
   if (status != 0){
     printf ("Fix_Reverse_X1 different id failed, exiting\n", status);
        return status;
  }

  status = Fix_Reverse_X2();
  if (status != 0){
           printf("Fix_Reverse_X2 failed, exiting\n");
return status;
  }

  status = Fix_Reverse_diffID_X2();
  if (status != 0){
           printf("Fix_Reverse_X2 different ID failed, exiting\n");
     return status;
  }

  printf("Please enter balance level ");
scanf("%d",&Balancelevel);

  while (Balancelevel >2) {
           printf("The value is not valid,please enter an other integer less than 3 ");
     scanf("%d",&Balancelevel);
  }
  status = balance(Balancelevel);
  if (status != 0){
           printf("balance failed, exiting\n");
```

```
return status;
  }

  status = CPXwriteprob(env, lp, "1979.lp", NULL);
  if (status != 0){
          printf ("Failed to write problem %d.\n", status);
        return status;
  }

  PList = PenaltyList;
  fprintf(file1,"*******Start of Penalty List******\n");
  while (PList != NULL){
          fprintf(file1,"ILogIdx=%d\n",PList->ILogIdx);
    fprintf(file1,"Sen=%d\n",PList->Sen);
          fprintf(file1,"Prior=%d\n",PList->Prior);
          fprintf(file1,"RecType=%c\n\n",PList->RecType);
          PList=PList->RightLink;
  }
  fprintf(file1,"*******End of Penalty List******\n");
PList=NULL;

  fclose(file1);

  PList = PenaltyList;
  if (PList == NULL){
    printf("Error, empty penalty list\n");
return -1;
  }
  while (PList->RightLink != NULL)
    PList = PList->RightLink;

  while (PList != NULL)
        PList = DoPenalty(PList);


  status = CPXwriteprob(env, lp, "1979.lp", NULL);
  if (status != 0){
```

```
            printf ("Failed to write problem %d.\n", status);
          return status;
  }

  status = solve_evaluate_results();
  if (status != 0){
          printf("Evaluate results failed, exiting\n");
          return status;
  }

  stop = clock();

  status = free_problem();
  if (status != 0){
           printf("free_problem failed, exiting\n");
     return status;
  }

  duration = (long double) (stop-start)/CLOCKS_PER_SEC;
  printf("Time = %Lf seconds\n", duration);

  printf("Press return to continue...\n");
getchar();

} //End main
```

ΠΑΡΑΡΤΗΜΑ Β’

Στο Παράρτημα Β’ παραθέτουμε τους περιορισμούς ισορροπίας του προβλήματος 2 του 6ου κεφαλαίου. Αρχικά παρουσιάζονται οι περιορισμοί ισορροπίας για balancelevel = 1 και έπειτα για balancelevel = 2.

balance level = 1

balance_constraintPOS0SES1SES2: x1 - x2 + x3 + x7 - x8 - x13 + x16 - x17 + x25 + x28 - x29 - x35 + x53 - x54 = 0

balance_constraintPOS0SES1SES3: x1 + x3 + x7 + x16 - x19 - x22 + x25 + x28 - x33 - x36 + x55 - x56 = 0

balance_constraintPOS0SES1SES4: x1 + x3 - x4 + x7 - x9 - x11 - x14 + x16 - x18 - x21 - x23 + x25 - x26 - x27 + x28 - x31 + x57 - x58 = 0

balance_constraintPOS0SES1SES5: x1 + x3 + x7 - x12 + x16 - x20 + x25 + x28 - x30 - x32 + x59 - x60 = 0

balance_constraintPOS0SES1SES6: x1 + x3 - x5 - x6 + x7 - x10 - x15 + x16 - x24 + x25 + x28 - x34 + x61 - x62 = 0

balance_constraintPOS0SES2SES3: x2 + x8 + x13 + x17 - x19 - x22 + x29 - x33 + x35 - x36 + x63 - x64 = 0

balance_constraintPOS0SES2SES4: x2 - x4 + x8 - x9 - x11 + x13 - x14 + x17 - x18 - x21 - x23 - x26 - x27 + x29 - x31 + x35 + x65 - x66 = 0

balance_constraintPOS0SES2SES5: x2 + x8 - x12 + x13 + x17 - x20 + x29 - x30 - x32 + x35 + x67 - x68 = 0

balance_constraintPOS0SES2SES6: x2 - x5 - x6 + x8 - x10 + x13 - x15 + x17 - x24 + x29 - x34 + x35 + x69 - x70 = 0

balance_constraintPOS0SES3SES4: - x4 - x9 - x11 - x14 - x18 + x19 - x21 + x22 - x23 - x26 - x27 - x31 + x33 + x36 + x71 - x72 = 0

balance_constraintPOS0SES3SES5: - x12 + x19 - x20 + x22 - x30 - x32 + x33 + x36 + x73 - x74 = 0

balance_constraintPOS0SES3SES6: - x5 - x6 - x10 - x15 + x19 + x22 - x24 + x33 - x34 + x36 + x75 - x76 = 0

balance_constraintPOS0SES4SES5: x4 + x9 + x11 - x12 + x14 + x18 - x20 + x21 + x23 + x26 + x27 - x30 + x31 - x32 + x77

- x78 = 0

balance_constraintPOS0SES4SES6: x4 - x5 - x6 + x9 - x10 + x11 + x14 - x15 + x18 + x21 + x23 - x24 + x26 + x27 + x31 - x34 + x79 - x80 = 0

balance_constraintPOS0SES5SES6: - x5 - x6 - x10 + x12 - x15 + x20 - x24 + x30 + x32 - x34 + x81 - x82 = 0

balance_constraintPOS1SES1SES2: x1 - x2 + x3 + x16 - x17 + x25 - x35 + x83 - x84 = 0

balance_constraintPOS1SES1SES3: x1 + x3 + x16 - x19 - x22 + x25 - x33 - x36 + x85 - x86 = 0

balance_constraintPOS1SES1SES4: x1 + x3 - x4 - x9 + x16 - x18 - x21 - x23 + x25 - x26 - x27 - x31 + x87 - x88 = 0

balance_constraintPOS1SES1SES5: x1 + x3 + x16 + x25 - x32 + x89 - x90 = 0

balance_constraintPOS1SES1SES6: x1 + x3 - x6 - x10 + x16 + x25 - x34 + x91 - x92 = 0

balance_constraintPOS1SES2SES3: x2 + x17 - x19 - x22 - x33 + x35 - x36 + x93 - x94 = 0

balance_constraintPOS1SES2SES4: x2 - x4 - x9 + x17 - x18 - x21 - x23 - x26 - x27 - x31 + x35 + x95 - x96 = 0

balance_constraintPOS1SES2SES5: x2 + x17 - x32 + x35 + x97 - x98 = 0

balance_constraintPOS1SES2SES6: x2 - x6 - x10 + x17 - x34 + x35 + x99 - x100 = 0

balance_constraintPOS1SES3SES4: - x4 - x9 - x18 + x19 - x21 + x22 - x23 - x26 - x27 - x31 + x33 + x36 + x101 - x102 = 0

balance_constraintPOS1SES3SES5: x19 + x22 - x32 + x33 + x36 + x103 - x104 = 0

balance_constraintPOS1SES3SES6: - x6 - x10 + x19 + x22 + x33 - x34 + x36 + x105 - x106 = 0

balance_constraintPOS1SES4SES5: x4 + x9 + x18 + x21 + x23 + x26 + x27 + x31 - x32 + x107 - x108 = 0

balance_constraintPOS1SES4SES6: x4 - x6 + x9 - x10 + x18 + x21 + x23 + x26 + x27 + x31 - x34 + x109 - x110 = 0

balance_constraintPOS1SES5SES6: - x6 - x10 + x32 - x34 + x111 - x112 = 0

balance_constraintPOS2SES1SES2: x7 - x8 - x13 + x28 - x29 + x113 - x114 = 0

balance_constraintPOS2SES1SES4: x7 - x11 - x14 + x28 + x115 - x116 = 0

balance_constraintPOS2SES1SES5: x7 - x12 - x20 + x28 - x30 + x117 - x118 = 0

balance_constraintPOS2SES1SES6: - x5 + x7 - x15 - x24 + x28 + x119 - x120 = 0

balance_constraintPOS2SES2SES4: x8 - x11 + x13 - x14 + x29 + x121 - x122 = 0

balance_constraintPOS2SES2SES5: x8 - x12 + x13 - x20 + x29 - x30 + x123 - x124 = 0

balance_constraintPOS2SES2SES6: - x5 + x8 + x13 - x15 - x24 + x29 + x125 - x126 = 0

balance_constraintPOS2SES4SES5: x11 - x12 + x14 - x20 - x30 + x127 - x128 = 0

balance_constraintPOS2SES4SES6: - x5 + x11 + x14 - x15 - x24 + x129 - x130 = 0

balance_constraintPOS2SES5SES6: - x5 + x12 - x15 + x20 - x24 + x30 + x131- x132=0

balance level = 2

balance_constraintPOS0SES1SES2: x1 - x2 + x3 + x7 - x8 - x13 + x16 - x17 + x25 + x28 - x29 - x35 + x53 - x54 = 0

balance_constraintPOS0SES1SES3: x1 + x3 + x7 + x16 - x19 - x22 + x25 + x28 - x33 - x36 + x55 - x56 = 0

balance_constraintPOS0SES1SES4: x1 + x3 - x4 + x7 - x9 - x11 - x14 + x16 - x18 - x21 - x23 + x25 - x26 - x27 + x28 - x31 + x57 - x58 = 0

balance_constraintPOS0SES1SES5: x1 + x3 + x7 - x12 + x16 - x20 + x25 + x28 - x30 - x32 + x59 - x60 = 0

balance_constraintPOS0SES1SES6: x1 + x3 - x5 - x6 + x7 - x10 - x15 + x16 - x24 + x25 + x28 - x34 + x61 - x62 = 0

balance_constraintPOS0SES2SES3: x2 + x8 + x13 + x17 - x19 - x22 + x29 - x33 + x35 - x36 + x63 - x64 = 0

balance_constraintPOS0SES2SES4: x2 - x4 + x8 - x9 - x11 + x13 - x14 + x17 - x18 - x21 - x23 - x26 - x27 + x29 - x31 + x35 + x65 - x66 = 0

balance_constraintPOS0SES2SES5: x2 + x8 - x12 + x13 + x17 - x20 + x29 - x30 - x32 + x35 + x67 - x68 = 0

balance_constraintPOS0SES2SES6: x2 - x5 - x6 + x8 - x10 + x13 - x15 + x17 - x24 + x29 - x34 + x35 + x69 - x70 = 0

balance_constraintPOS0SES3SES4: - x4 - x9 - x11 - x14 - x18 + x19 - x21 + x22 - x23 - x26 - x27 - x31 + x33 + x36 + x71 - x72 = 0

balance_constraintPOS0SES3SES5: - x12 + x19 - x20 + x22 - x30 - x32 + x33 + x36 + x73 - x74 = 0

balance_constraintPOS0SES3SES6: - x5 - x6 - x10 - x15 + x19 + x22 - x24 + x33 - x34 + x36 + x75 - x76 = 0

balance_constraintPOS0SES4SES5: x4 + x9 + x11 - x12 + x14 + x18 - x20 + x21 + x23 + x26 + x27 - x30 + x31 - x32 + x77 - x78 = 0

balance_constraintPOS0SES4SES6: x4 - x5 - x6 + x9 - x10 + x11 + x14 - x15 + x18 + x21 + x23 - x24 + x26 + x27 + x31 - x34 + x79 - x80 = 0

balance_constraintPOS0SES5SES6: - x5 - x6 - x10 + x12 - x15 + x20 - x24 + x30 + x32 - x34 + x81 - x82 = 0

balance_constraintPOS1SES1SES2: x1 - x2 + x16 - x17 - x35 + x83 - x84 = 0

balance_constraintPOS1SES1SES3: x1 + x16 - x36 + x85 - x86 = 0

balance_constraintPOS1SES1SES4: x1 - x9 + x16 - x18 - x27 - x31 + x87 - x88 = 0

balance_constraintPOS1SES1SES5: x1 + x16 - x32 + x89 - x90 = 0

balance_constraintPOS1SES1SES6: x1 - x6 - x10 + x16 + x91 - x92 = 0

balance_constraintPOS1SES2SES3: x2 + x17 + x35 - x36 + x93 - x94 = 0

balance_constraintPOS1SES2SES4: x2 - x9 + x17 - x18 - x27 - x31 + x35 + x95 - x96 = 0

balance_constraintPOS1SES2SES5: x2 + x17 - x32 + x35 + x97 - x98 = 0

balance_constraintPOS1SES2SES6: x2 - x6 - x10 + x17 + x35 + x99 - x100 = 0

balance_constraintPOS1SES3SES4: - x9 - x18 - x27 - x31 + x36 + x101 - x102 = 0

balance_constraintPOS1SES3SES5: - x32 + x36 + x103 - x104 = 0

balance_constraintPOS1SES3SES6: - x6 - x10 + x36 + x105 - x106 = 0

balance_constraintPOS1SES4SES5: x9 + x18 + x27 + x31 - x32 + x107 - x108 = 0

balance_constraintPOS1SES4SES6: - x6 + x9 - x10 + x18 + x27 + x31 + x109 - x110 = 0

balance_constraintPOS1SES5SES6: - x6 - x10 + x32 + x111 - x112 = 0

balance_constraintPOS2SES1SES3: x3 - x19 - x22 + x25 - x33 + x113 - x114 = 0

balance_constraintPOS2SES1SES4: x3 - x4 - x21 - x23 + x25 - x26 + x115 - x116 = 0

balance_constraintPOS2SES1SES6: x3 + x25 - x34 + x117 - x118 = 0

balance_constraintPOS2SES3SES4: - x4 + x19 - x21 + x22 - x23 - x26 + x33 + x119 - x120 = 0

balance_constraintPOS2SES3SES6: x19 + x22 + x33 - x34 + x121 - x122 = 0

balance_constraintPOS2SES4SES6: x4 + x21 + x23 + x26 - x34 + x123 - x124 = 0

balance_constraintPOS3SES1SES2: x7 - x8 - x13 + x28 - x29 + x125 - x126 = 0

balance_constraintPOS3SES1SES4: x7 - x11 - x14 + x28 + x127 - x128 = 0

balance_constraintPOS3SES1SES5: x7 - x12 - x20 + x28 - x30 + x129 - x130 = 0

balance_constraintPOS3SES1SES6: - x5 + x7 - x15 - x24 + x28 + x131 - x132 = 0

balance_constraintPOS3SES2SES4: x8 - x11 + x13 - x14 + x29 + x133 - x134 = 0

balance_constraintPOS3SES2SES5: x8 - x12 + x13 - x20 + x29 - x30 + x135 - x136 = 0

balance_constraintPOS3SES2SES6: - x5 + x8 + x13 - x15 - x24 + x29 + x137 - x138 = 0

balance_constraintPOS3SES4SES5: x11 - x12 + x14 - x20 - x30 + x139 - x140 = 0

balance_constraintPOS3SES4SES6: - x5 + x11 + x14 - x15 - x24 + x141 - x142 = 0

balance_constraintPOS3SES5SES6: - x5 + x12 - x15 + x20 - x24 + x30 + x143 - x144 = 0

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] M. Shapiro, "Scheduling Crewmen for Recurrent Training," *Interfaces, 11 (3),* pp. 1-8, 1981.

[2] R. P. Brown, "Optimizing Readiness and Equity in Marine Corps Aviation Training Schedules," *Master thesis, Naval Postgraduate School at Monterey, California, USA,* 1995.

[3] M. I. Hall, "Optimal Scheduling of Army Initial Entry Training Courses," *Master thesis, Naval Postgraduate School at Monterey, California, USA,* 1999.

[4] G. Yu, S. Dugan and M. Argüello, "Moving Toward an Integrated Decision Support System for Manpower Planning at Continental Airlines: Optimization of Pilot Training Assignments," *Industrial Applications of Combinatorial Optimization, Applied Optimization, 16,* pp. 1-24, 1998.

[5] B. G. Thengvall and X. Qi, "System and Method for Rapid Generation of Minimum Length Pilot Training Schedules". United States Patent 0139958, 2003.

[6] M. G. Sohoni, T. G. Bailey, K. G. Martin, H. Carter and E. L. Johnson, "Delta Optimizes Continuing-Qualification-Traing Schedules for Pilots," *Interfaces, 33 (5),* pp. 57-70, 2003.

[7] G. Yu, J. Pachon, B. Thengvall, D. Chandler and A. Wilson, "Optimizing Pilot Planning and Training for Continental Airlines," *Interfaces, 34 (4),* pp. 253-264, 2004.

[8] X. Qi, J. F. Bard and G. Yu, "Class Scheduling for Pilot Training," *Operations Research, 52 (1),* pp. 148-162, 2004.

[9] J. Xu, M. Sohoni, M. McCleery and T. G. Bailey, "A dynamic neighborhood based tabu search algorith for real-world flight instructor scheduling problems," *European Journal of Operational Research,169,* pp. 978-993, 2006.

[10]B. G. Thengvall and J. E. Pachon, "Integrated Decision Support System for Optimizing the Training and Transition of Airline Pilots". United States Patent 7,346,528 B2, 2008.

[11] R. Hábel, "Pilot Training Optimization for Airlines," *Master thesis, Eötvös Loránd University, Budapest, Hungary,* 2013.

[12] Aslan Davut, "*A decision support system for effective scheduling in an F-16 pilot training squadron*'', Master Thesis, Air Force Inst. of Tech., Wright-Patterson AFB,

OH. SchoolofEngineeringandManagement.

[13] °Asa Holm, ‘’ *Manpower Planning in Airlines - Modeling and Optimization*’’,Master thesis, Link¨opings Universitet,2008

[14] J. Ryan McLaughlin,’’*OPTIMIZING ADVERSARY TRAINING AND THE STRUCTURE OF THE NAVY ADVERSARY FLEET*’’,Master thesis,Naval Postgraduate school,California 2010

[14] Πανταζή θεοδώρα, ‘*Μεικτός Ακέραιος Προγραμματισμός για βέλτιστη ανάθεση ιπτάμενων σε εκπαιδευτικές συνεδρίες προσομοιωτή’*, Διπλωματική εργασία, Τμήμα Μηχανολόγων Μηχανικών, Πανεπιστήμιο Θεσσαλίας,2013.

[15]Π.Χ.Γ. Παναγιώτου και Ν.Δ. Τσάντας Εισαγωγή στην Επιχειρισιακή Έρευνα, Εκδόσεις Ζήτη, Θεσσαλονίκη, 2000.

[16] S. Gass, A.Assad, “ *An annotated timeline of Operational Research, an informal history*’’, Kluwer Academic Publishers, c2005.