**ΙΩΑΝΝΗΣ ΚΑΠΕΤΑΝΓΕΩΡΓΗΣ** **Α.Μ. 1115201800061**

# PROJECT 3
# Τεχνητή Νοημοσύνη

## Πρόβλημα 1 - RLFA

### Αρχεία κώδικα:

Τα αρχεία που περιέχονται στον φάκελο είναι:
- csp.py: Στο αρχείο αυτό περιέχονται οι υλοποιήσεις των αλγορίθμων. Το αρχείο αυτό περιέχει τον κώδικα από το aima, με κάποιες επιπλέον προσθήκες έτσι ώστε να υλοποιηθούν όλα όσα ζητούνται από την άσκηση. Οι προσθήκες αυτές περιγράφονται αναλυτικά παρακάτω.
- utils.py, search.py: Τα αρχεία αυτά περιέχουν ήδη υλοποιημένες συναρτήσεις του aima οι οποίες χρησιμοποιούνται από το csp.py. Στα αρχεία αυτά δεν έχω προσθέσει/τροποποιήσει κάτι. Η μόνη διαφορά είναι ότι έχω κρατήσει μόνο όσες συναρτήσεις χρησιμοποιούνται, καθώς στα original αρχεία του aima υπάρχουν πολύ περισσότερες συναρτήσεις τις οποίες δεν χρειαζόμαστε.
- readFiles.py: Στο αρχείο αυτό περιέχονται οι συναρτήσεις τις οποίες υλοποίησα και χρησιμοποιούνται έτσι ώστε να διαβαστούν τα variables, τα domains και τα constraints από τα αρχεία που μας δόθηκαν.
- main.py: Το αρχείο αυτό είναι και το αρχείο το οποίο εκτελούμε. Οι λειτουργίες που περιέχει εξηγούνται παρακάτω αναλυτικά.

Ο κώδικας είναι πλήρως σχολιασμένος και εξηγείται πολύ αναλυτικά η κάθε λειτουργία.

**Παρατήρηση:**
Τα .txt αρχεία τα οποία περιέχουν τα constraints, τα domains και τα variables πρέπει να περιέχονται όλα σε έναν φάκελο με όνομα rlfap ο οποίο βρίσκεται στο ίδιο directory με το αρχείο readFiles.py.

### Εκτέλεση προγράμματος

Για να εκτελέσουμε το πρόγραμμα χρειάζεται να εκτελέσουμε το αρχείο main.py. Επίσης χρειάζεται να δώσουμε δύο επιπλέον ορίσματα τα οποία αφορούν το ποιος αλγόριθμος θα εκτελεστεί και για ποιό instance. Πιο συγκεκριμένα, το πρώτο όρισμα πρέπει να είναι ένα από τα ακόλουθα: **fc** , **mac** , **fc-cbj** , **min-conf** όπου το fc αντιστοιχεί στον αλγόριθμο forward-checking, το mac στον MAC το fc-cbj στον FC-CBJ και το min-conf στον Min Conflicts. Το δεύτερο όρισμα πρέπει να περιέχει μόνο το όνομα του instance, δηλαδή να είναι ένα από τα ακόλουθα: **11** , **2-f24** , **2-f25** , **3-f10** , **8-f10** , **8-f11** , **14-f27** , **14-f28** , **6-w2** , **7-w1-f4** , **7-w1-f5 .**
Ένα παράδειγμα εκτέλεσης είναι το εξής: **python3 fc-cbj 14-f27**

### readFiles.py - Μοντελοποίηση προβλήματος:

Στο αρχείο αυτό περιέχονται οι συναρτήσεις με τις οποίες διαβάζονται τα δεδομένα του προβλήματος από τα αρχεία. Πιο συγκεκριμένα υπάρχουν 3 συναρτήσεις (μία για κάθε αρχείο). Οι συναρτήσεις αυτές είναι:
- readVars: Η συνάρτηση αυτή διαβάζει τα δεδομένα των αρχείων var*.txt. Δημιουργεί και επιστρέφει ένα dictionary το οποίο έχει σαν κλειδιά τις μεταβλητές και σαν τιμές το domain που ανήκει η κάθε μεταβλητή-κλειδί.
- readDoms: Η συνάρτηση αυτή διαβάζει τα δεδομένα των αρχείων dom*.txt. Δημιουργεί και επιστρέφει ένα dictionary το οποίο έχει σαν κλειδιά τα domains και σαν τιμή για κάθε κλειδί μια λίστα με τις τιμές του κάθε domain.

Έπειτα από την κλήση των 2 αυτών συναρτήσεων, στην main μέσω μιας επαναληπτικής διαδικασίας αντιστοιχίζονται οι τιμές των δύο αυτών dictionaries. Πιο συγκεκριμένα δημιουργείται ένα νέο dictionary το οποίο έχει σαν κλειδιά τις μεταβλητές και σαν τιμή για κάθε κλειδί μια λίστα με τις αντίστοιχες τιμές που μπορεί να πάρει η μεταβλητή αυτή (σύμφωνα με το domain στο οποίο ανήκει).
Παίρνοντας μόνο τα κλειδιά του dictionary αυτού, έχουμε και μια λίστα με της μεταβλητές του προβλήματος.
- readCtrs: Η συνάρτηση αυτή διαβάζει τα δεδομένα των αρχείων ctr*.txt. Δημιουργεί και επιστρέφει ένα dictionary το οποίο έχει σαν κλειδί ένα tuple της μορφής (variable1,variable2) όπου variable1 και variable2 είναι οι δύο μεταβλητές που “συμμετέχουν” στον περιορισμό αυτό. Η τιμή που αντιστοιχεί σε κάθε κλειδί είναι ένα tuple το οποίο περιέχει τον τελεστή του περιορισμού (δηλαδή =,<,>) και τον αριθμό ο οποίος βρίσκεται στο δεξιό μέλος του περιορισμού. Επίσης η συνάρτηση δημιουργεί ακόμα ένα λεξικό με τους neighbors της κάθε μεταβλητής. Το dictionary αυτό έχει σαν κλειδιά τις μεταβλητές και σαν τιμή που αντιστοιχεί σε κάθε κλειδί μια λίστα με τους γείτονες της. Η διαδικασία που ακολουθείται για την εύρεση των γειτόνων είναι η εξής: Κάθε constraint το οποίο διαβάζεται από το αρχείο αφορά δύο μεταβλητές (έστω variable1 και variable2). Έτσι προσθέτουμε την variable2 στους γείτονες της variable1 και αντίστοιχα προσθέτουμε την variable1 στους γείτονες της variable2. Αυτό γίνεται για κάθε ένα constraint που διαβάζεται.

Έπειτα από όλα τα παραπάνω, τελικά έχουμε: τις μεταβλητές, τα domains των μεταβλητών, τους γείτονες των μεταβλητών και τα constraints.
Δηλαδή έχουμε μοντελοποιήσει κατάλληλα το πρόβλημα μας.

## main.py

Στο αρχείο αυτό αρχικά καλούνται οι συναρτήσεις που εξηγήθηκαν στην παραπάνω παράγραφο έτσι ώστε να διαβάσουμε τα δεδομένα του προβλήματος. Μέσω της παραμέτρου που δόθηκε κατά την εκτέλεση του προγράμματος, δίνουμε το αντίστοιχο όρισμα στις συναρτήσεις έτσι ώστε να ξέρουν από ποια αρχεία να διαβάσουν τα δεδομένα.
Έπειτα από την κλήση των συναρτήσεων και την εκτέλεση των λειτουργιών που περιγράφηκαν παραπάνω, έχουμε δημιουργήσει όλες τις δομές οι οποίες περιέχουν τα δεδομένα του προβλήματος.
Στην συνέχεια ορίζουμε την συνάρτηση μέσω τις οποίας θα ελέγχονται οι περιορισμοί. Η συνάρτηση αυτή δέχεται σαν ορίσματα δύο μεταβλητές και τις αντίστοιχες τιμές που έχουν ανατεθεί στην κάθε μία. Έτσι ψάχνουμε αν υπάρχει κάποιος περιορισμός που αφορά τις 2 αυτές μεταβλητές (στην δομή που περιέχει τους περιορισμούς η οποία μας επιστράφηκε από την συνάρτηση readCtrs), και αν υπάρχει ελέγχουμε αν αυτός τηρείται. Περαιτέρω λεπτομέρειες εξηγούνται στα σχόλια του κώδικα.
Έχοντας υλοποιήσει και την συνάρτηση αυτή, έχουμε όσα χρειαζόμαστε για να ορίσουμε το πρόβλημα.
Έτσι μέσω της κλάσης csp του αρχείου csp.py δημιουργούμε το πρόβλημα μας δίνοντας σαν ορίσματα για την αρχικοποίηση: τις μεταβλητές, τα domain των μεταβλητών, του γείτονες των μεταβλητών, τα constraints και την συνάρτηση η οποία ελέγχει τα constraints.
Παρατήρηση: τα constraints έχουν προστεθεί σαν όρισμα καθώς χρειάζονται για την υλοποίηση του dom_wdeg. Περισσότερες λεπτομέρειες εξηγούνται παρακάτω.

Έχοντας ορίσει το πρόβλημα, το μόνο που απομένει είναι να καλέσουμε τον κατάλληλο αλγόριθμο. Με βάση της παραμέτρου που δόθηκε κατά την κλήση του προγράμματος καλούμε και τον αντίστοιχο αλγόριθμο.
- Για τον fc καλείται ο backtracking_search με variable-ordering συνάρτηση την DOM/WDEG , value-ordering συνάρτηση την lcv και inference προφανώς την forward_checking.
- Για τον mac καλείται ο backtracking_search με variable-ordering συνάρτηση την DOM/WDEG , value-ordering συνάρτηση την lcv και inference προφανώς την MAC.
- Για τον FC-CBJ καλείται ο cbj_search με variable-ordering συνάρτηση την DOM/WDEG , value-ordering συνάρτηση την lcv και inference προφανώς την forward_checking.
- Για τον Min-Conflicts καλείται ο min_conflicts, είτε δίνοντας του σαν όρισμα τον μέγιστο αριθμό βημάτων είτε χωρίς επιπλέον όρισμα (δηλαδή ο μέγιστος αριθμός βημάτων καθορίζεται από την default τιμή του αλγορίθμου που ορίζεται στο αρχείο csp.py)

Έπειτα από την εκτέλεση του αλγορίθμου εκτυπώνεται η λύση του προβλήματος (ή None αν δεν υπάρχει λύση) και στην συνέχεια ο αριθμός των αναθέσεων που έγιναν κατά την εκτέλεση του αλγορίθμου και ο αριθμός των ελέγχων που έγιναν από την inference συνάρτηση (για τους τρεις πρώτους αλγορίθμους).

## csp.py

Το αρχείο αυτό περιέχει τον κώδικα του aima που μας δόθηκε, με κάποιες επιπλέον προσθήκες/τροποποιήσεις. Επίσης έχουν αφαιρεθεί ορισμένα κομμάτια κώδικα τα οποία δεν μας χρησιμεύουν, όπως τα ενδεικτικά προβλήματα που περιείχε το original αρχείο του aima.
Οι τροποποιήσεις/προσθήκες είναι οι εξής:

### Κλάση CSP:
Στα αντικείμενα της κλάσης CSP έχουν προστεθεί κάποια επιπλέον χαρακτηριστικά. Πιο συγκεκριμένα έχουν προστεθεί:
- **weight**: Ένα dictionary το οποίο περιέχει το βάρος του κάθε constraint το οποίο χρειαζόμαστε για την υλοποίηση της ευρετικής συνάρτησης DOM/WDEG. Το dictionary αυτό έχει ως κλειδιά το κάθε constraint (δηλαδή ένα tuple με τις δύο μεταβλητές τις οποίες αφορά το constraint). Η τιμή που αντιστοιχεί στο κάθε κλειδί/constraint είναι και το βάρος του. Τα βάρη αρχικοποιούνται με την τιμή 1 (περαιτέρω λεπτομέριες για την DOM/WDEG και τα βάρη εξηγούνται παρακάτω). Για να μπορέσουμε να δημιουργήσουμε το dictionary αυτό χρειαζόμαστε τα constraints. Έτσι, έχει προστεθεί ένα επιπλέον όρισμα στον initializer της κλάσης CSP το οποίο είναι το dictionary με τα constraints που εξηγήθηκε και παραπάνω.
- **conflict_set**: Ένα dictionary το οποίο περιέχει το conflict set της κάθε μεταβλητής και χρησιμοποιείται από τον αλγόριθμο cbj. Προφανώς έχει ως κλειδιά τις μεταβλητές και ως τιμή για κάθε κλειδί μια λίστα από μεταβλητές. Επίσης έχει προστεθεί ακόμα ένα χαρακτηριστικό, το deadend_conflict_set το οποίο χρησιμεύει έτσι ώστε να αποθηκεύουμε το conflict set μιας συγκεκριμένης μεταβλητής (περισσότερες λεπτομέρειες παρακάτω).
- **deadend**: Στην μεταβλητή αυτή αποθηκεύεται η πληροφορία για το αν έχουμε φτάσει σε deadend ή όχι και χρησιμοποιείται από τον αλγόριθμο cbj. Αν έχουμε φτάσει σε deadend, τότε η μεταβλητή deadend περιέχει την μεταβλητή στην οποία συναντήσαμε το αδιέξοδο, αλλιώς η μεταβλητή deadend περιέχει None.
- **checksCount**: Περιέχει τον συνολικό αριθμό ελέγχων που έγιναν από έναν αλγόριθμο ελέγχου περιορισμών (π.χ. forward_checking). Κάθε φορά που καλείται η συνάρτηση ελέγχου περιορισμών ο αριθμός ελέγχων που έκανε κατά την συγκεκριμένη κλήση προστίθεται στον συνολικό αριθμό ελέγχων checksCount. Τον συνολικό αριθμό ελέγχων  τον χρησιμοποιούμε σαν μετρική σύγκρισης των αλγορίθμων.

## Ευρετική συνάρτηση - DOM/WDEG

Στην συνάρτηση αυτή, αρχικά, πρέπει να αναθέτουμε ένα βάρος για κάθε ένα constraint. Έτσι κατά της αρχικοποίηση του προβλήματος δημιουργούμε ένα dictionary (με όνομα weight) το οποίο έχει ως κλειδιά τα constraints και ως τιμές το βάρος που αντιστοιχεί στο κάθε κλειδί constraint. Το βάρος του κάθε constraint αρχικοποιείται με 1 κατά την αρχικοποίηση του προβλήματος (δηλαδή από τον initializer της κλάσης csp).

Στην συνέχεια πρέπει να προσθέσουμε το πότε αυξάνεται το βάρος του κάθε constraint. Η αύξηση του βάρους γίνεται εσωτερικά της κλήσης της συνάρτησης ελέγχου περιορισμών (inference). Στην περίπτωση μας χρησιμοποιούμε δύο τέτοιες συναρτήσεις: την forward_checking και την MAC (η οποία χρησιμοποιεί την AC3). Οπότε:

- Για την forward_checking: Κάθε φορά που κατά τον έλεγχο των περιορισμών εσωτερικά της forward_checking και έπειτα από το pruning, ένας γείτονας της μεταβλητής της οποίας μόλις της ανατέθηκε μια νέα τιμή μείνει με “άδειο domain” τότε πριν η συνάρτηση επιστρέψει false αυξάνουμε κατά 1 το βάρος του περιορισμού μεταξύ των δύο αυτών μεταβλητών που οδήγησε στο άδειο domain του γείτονα. (line 447 αρχείο csp.py)
- Ο MAC χρησιμοποιεί την AC3. Η AC3 καλεί την συνάρτηση revice έτσι ώστε να ελεγχθεί η συνέπεια των ακμών. Επομένως, η αύξηση του βάρους πρέπει να γίνει εσωτερικά της συνάρτησης revice. Πιο συγκεκριμένα, έπειτα από τον έλεγχο των περιορισμών και το pruning του domain της μεταβλητής Xi, εάν το domain της Xi τελικά έμεινε άδειο (δηλαδή όλες οι τιμές έγιναν pruned) τότε αυξάνουμε το βάρος του περιορισμού που οδήγησε στο άδειο domain, δηλαδή του περιορισμού μεταξύ των δύο μεταβλητών που ελέγχαμε Xi και Xj. (line 224 αρχείο csp.py)

Το μόνο που μένει είναι η υλοποίηση της συνάρτησης η οποία θα επιλέγει κάθε φορά την κατάλληλη μεταβλητή η οποία θα γίνει assigned στην συνέχεια του αλγορίθμου. Η συνάρτηση αυτή έχει όνομα dom_wdeg και η υλοποίηση της είναι στην γραμμή 386 του αρχείου csp.py.
Η διαδικασία που ακολουθείται είναι:

- Για κάθε μία unassigned μεταβλητή (var1) βρίσκουμε όλα τα constraints τα οποία περιέχουν την μεταβλητή αυτή καθώς και μία ακόμα unassigned μεταβλητή (var2).
- Αν δεν υπάρχει κανένα τέτοιο contraint τότε συνεχίζουμε με την επόμενη μεταβλητή καθώς δεν μπορεί να υπολογιστεί το πηλίκο dom/wdeg. Αν υπάρχει τουλάχιστον ένα τέτοιο constraint τότε αθροίζουμε τα βάρη όλων των ακμών που πληρούν τις προϋποθέσεις αυτές.
- Στην συνέχεια βρίσκουμε το πλήθος των τιμών που περιέχονται στο domain της μεταβλητής var1.
- Έτσι έχουμε υπολογίσει όλα όσα χρειαζόμαστε για να υπολογίσουμε το πηλίκο dom/wdeg, το οποίο και υπολογίζουμε.
- Τελικά κρατάμε την μεταβλητή η οποία έχει το μικρότερο πηλίκο την οποία και επιστρέφουμε.
- Στην περίπτωση που δεν μπορούμε να υπολογίσουμε το πηλίκο για καμία από τις μεταβλητές, τότε επιστρέφουμε την πρώτη unassigned μεταβλητή καλώντας την συνάρτηση first_unassigned_variable.

Η συνάρτηση αυτή χρησιμοποιείται σαν variable-ordering συνάρτηση από τους αλγορίθμους backtracking_search και cbj_search. Αυτό γίνεται δίνοντας την σαν όρισμα κατά την κλήση των δύο τελευταίων.
**Παρατήρηση:** Η dom/wdeg δεν περιέχει κανέναν παράγοντα τυχαιότητας, επομένως σε κάθε κλήση για το ίδιο πρόβλημα έχει την ίδια συμπεριφορά.
**Παρατήρηση 2:** Έχει υλοποιηθεί μια επιπλέον συνάρτηση με όνομα dom_wdeg_extra. Η συνάρτηση αυτή είναι ακριβώς ίδια με την dom_wdeg με την μόνη διαφορά ότι όταν δεν μπορεί να επιλέξει κάποια μεταβλητή χρησιμοποιεί την mrv αντί της first_unassigned_variable. Έτσι η dom_wdeg_extra περιέχει έναν παράγοντα τυχαιότητας. Η dom_wdeg_extra χρησιμοποιείται μόνο για την επιπλέον μελέτη που περιγράφεται μόνο στην τελευταία παράγραφο του προβλήματος 1. Η κανονική μελέτη έχει γίνει χρησιμοποιώντας την απλή dom_wdeg.

## Forward Checking - FC

Για την εκτέλεση του αλγορίθμου forward checking χρησιμοποιείται ο ήδη υλοποιημένος αλγόριθμος backtracking_search (ο οποίος έχει παραμείνει ίδιος με αυτόν του aima). Στον backtracking_search δίνουμε σαν όρισμα inference συνάρτησης την συνάρτηση forward_checking. Οι μόνες αλλαγές που έχουν γίνει στην συνάρτηση forward_checking είναι η αύξηση των βαρών που αφορούν την dom/wdeg και η προσθήκη ενός μετρητή ο οποίος μετρά τους ελέγχους περιορισμών που έγιναν σε κάθε κλήση της forward_checking. Πριν την ολοκλήρωση της κάθε κλήσης, ο αριθμός αυτός προστίθεται στον συνολικό αριθμό ελέγχων checksCount που είναι χαρακτηριστικό της κλάσης csp και εξηγήθηκε και παραπάνω.
Επίσης στον backtracking_search δίνονται τα ορίσματα dom_wdeg και lcv τα οποία αφορούν το variable-ordering και το value-ordering αντίστοιχα και προφανώς το πρόβλημα το οποίο θέλουμε να λύσουμε.
Άρα συνολικά η κλήση του αλγορίθμου για το forward_checking (όπως φαίνεται και στο αρχείο main.py) γίνεται ως εξής:

**backtracking_search(rlfsProblem, csp.dom_wedg, csp.lcv, csp.forward_checking)**

(όπου rlfsProblem το πρόβλημα που ορίζεται κάθε φορά)

## MAC

Για την εκτέλεση του αλγορίθμου MAC, χρησιμοποιείται επίσης ο αλγόριθμος backtracking_search. Σαν inference “δίνουμε" την συνάρτηση mac η οποία με την σειρά της καλεί την συνάρτηση του AC3. Η μόνες μετατροπές που έχουν γίνει είναι η αύξηση του βάρους για τον dom/wdeg η οποία ήδη εξηγήθηκε καθώς και ο αριθμός ελέγχων που κάνει κάθε φορά ο AC3. Ο αριθμός ελέγχων ήδη μετριόταν στην υλοποίηση του aima οπότε το μόνο που προστέθηκε είναι κάθε φορά να προσθέτουμε τον αριθμό αυτόν στον συνολικό αριθμό ελέγχων checksCount. Για τον MAC, ομοίως με τον FC, χρησιμοποιούμε dom/wdeg και lcv για variable-ordering και το value-ordering αντίστοιχα. Επομένως η κλήση του αλγορίθμου γίνεται ως εξής:

**backtracking_search(rlfsProblem, csp.dom_wedg, csp.lcv, csp.mac)**

## FC-CBJ

Για την υλοποίηση του αλγορίθμου FC-CBJ αρχικά υλοποίησα τον αλγόριθμο CBJ. Για τον CBJ βασίστηκα στην υλοποίηση του backtracking_search του aima κάνοντας τις απαραίτητες αλλαγές έτσι ώστε να μετατραπεί στον CBJ (βασιζόμενος και στο paper το οποίο μας δόθηκε). Πιο συγκεκριμένα, ο αλγόριθμός cbj_search έχει σαν υποσυνάρτηση την cbj η οποία εκτελείται αναδρομικά. Όπως και η backtracking_search έχει την συνάρτηση backtrack.
Επίσης, όπως έχει αναφερθεί και παραπάνω, για τον cbj χρειαζόμαστε κάποια επιπλέον δεδομένα. Τα δεδομένα αυτά είναι:

- Για κάθε μεταβλητή κρατάμε το conflict set της το οποίο περιέχει άλλες μεταβλητές οι οποίες έπειτα από μία ανάθεση τιμής αφαίρεσαν τιμές από το domain της μεταβλητής αυτής. Οι πληροφορίες αυτές κρατούνται σε ένα dictionary με όνομα conflict_set.
- Την πληροφορία για το αν έχουμε φτάσει σε ένα αδιέξοδο. Η πληροφορία αυτή αποθηκεύεται σε μια μεταβλητή με όνομα deadend. Η μεταβλητή αυτή περιέχει None αν δεν έχουμε φτάσει σε αδιέξοδο, ενώ αν έχουμε φτάσει σε αδιέξοδο περιέχει την μεταβλητή στην οποία φτάσαμε σε deadend. Σαν αδιέξοδο ορίζουμε όταν για μία μεταβλητή δεν υπάρχει καμία επιτρεπτή τιμή για να της αναθέσουμε. Η πληροφορία για το deadend μας χρησιμεύει έτσι ώστε να μπορούμε να κάνουμε το backjump στην κατάλληλη μεταβλητή.
- Μια μεταβλητή στην οποία αποθηκεύουμε το conflict set της μεταβλητής στην οποία καταλήξαμε σε deadend. Η μεταβλητή αυτή έχει όνομα deadend_conflict_set. Όπως γνωρίζουμε όταν κάνουμε ένα backjump πρέπει να προσθέσουμε στο conflict set της μεταβλητής στην οποία κάναμε jump, το conflict set της μεταβλητής στην οποία συναντήσαμε το αδιέξοδο. Αυτό το καταφέρνουμε μέσω της μεταβλητής deadend_conflict_set αφού έχουμε κάνει το backjump.

Οι προσθήκες που έγιναν είναι οι εξής:

- Έπειτα από την κάθε ανάθεση τιμής που γίνεται σε μία μεταβλητή καλείται η συνάρτηση updateConflictSet (της οποίας η υλοποίηση εξηγείται παρακάτω) έτσι ώστε να ανανεωθούν τα conflict sets των μεταβλητών των οποίων τα domains επηρεάζονται από την ανάθεση αυτή.
- Αν για μία μεταβλητή δεν υπάρχει καμία επιτρεπτή τιμή για να της αναθέσουμε τότε έχουμε φτάσει σε ένα αδιέξοδο (deadend). Στην περίπτωση αυτή χρειάζεται να κάνουμε ένα backjump. Έτσι, αναθέτουμε στην μεταβλητή deadend την τιμή της τρέχουσας μεταβλητής στην οποία φτάσαμε σε αδιέξοδο και επίσης αποθηκεύουμε το τρέχον conflict set της στην μεταβλητή deadend_conflict_set και επιστρέφουμε None έτσι ώστε να αρχίσει το backjumping.
- Αρχίζοντας να επιστρέφουμε προς τα πίσω από τις αναδρομικές κλήσεις, ελέγχουμε αν η τρέχουσα μεταβλητή που ελέγχεται στην συγκεκριμένη κλήση βρίσκεται στο conflict set της μεταβλητής στην οποία φτάσαμε σε αδιέξοδο (δηλαδή της μεταβλητής που περιέχεται στην deadend).
  - Αν υπάρχει τότε έχουμε καταφέρει να βρούμε την μεταβλητή του συνόλου συγκρούσεων της deadend μεταβλητής η οποία βρίσκεται βαθύτερα στο “δένδρο” αναζήτησης (αν αναπαραστήσουμε τις αναδρομικές κλήσεις σαν ένα δένδρο).
  - Αν δεν υπάρχει, τότε κάνουμε unassign την μεταβλητή αυτή και επιστρέφουμε None έτσι ώστε να συνεχίσουμε το backjumping έως ότου βρούμε την κατάλληλη μεταβλητή στην οποία πρέπει να κάνουμε το jump.
- Αφού “φτάσουμε” στην μεταβλητή που πρέπει να κάνουμε το backjump, πρέπει να προσθέσουμε τις μεταβλητές οι οποίες βρίσκονται στο conflict set της deadend μεταβλητής, στο conflict set της μεταβλητής στην οποία κάναμε backjump. Αυτό γίνεται καλώντας την συνάρτηση merge (της οποίας η λειτουργία εξηγείται παρακάτω).

updateConflictSet

Η συνάρτηση αυτή είναι υπεύθυνη για την ενημέρωση των conflict sets των μεταβλητών έπειτα από την ανάθεση τιμής σε μία μεταβλητή. Έστω ότι η μεταβλητή στην οποία ανατέθηκε μια τιμή είναι ονομάζεται var.
Για να υλοποιήσουμε το παραπάνω, έχουμε μια επαναληπτική διαδικασία η οποία:
Για κάθε γείτονα της μεταβλητής var (δηλαδή για κάθε μεταβλητή της οποίας το domain επηρεάζεται από την ανάθεση μιας τιμής στην μεταβλητή var), ελέγχεται αν η var υπάρχει ήδη στο conflict set του κάθε γείτονα και αν δεν υπάρχει τότε προστίθεται η μεταβλητή var.
Έτσι έχουμε καταφέρει να προσθέσουμε την μεταβλητή var στα conflict sets όλων των μεταβλητών οι οποίες επηρεάστηκαν από την ανάθεση της τιμής στην μεταβλητή var.

merge

Στην συνάρτηση αυτή γίνονται οι εξής 2 λειτουργίες:
- Αρχικά χρησιμοποιώντας την μεταβλητή deadend_conflict_set στην οποία είναι αποθηκευμένο το conflict set της μεταβλητής στην οποία καταλήξαμε σε deadend, προσθέτουμε τις μεταβλητές που περιέχει το deadend_conflict_set στο conflict set της μεταβλητής στην οποία κάναμε το backjump.
- Επίσης, έπειτα από το backjump κάποιες μεταβλητές έχουν γίνει unassigned. Έτσι αφαιρούμε τις μεταβλητές αυτές από όσα conflict set τις περιέχουν, αφού πλέον δεν επηρεάζουν τα domains των αντίστοιχων μεταβλητών (καθώς πλέον δεν έχουν κάποια τιμή).

Συνολικά:
Πέρα από τις παραπάνω προσθήκες οι υπόλοιπες λειτουργίες καθώς και η δομή της μεθόδου είναι ακριβώς ίδια με τον backtracking_search του aima.
Άρα συνολικά έχουμε καταφέρει να υλοποιήσουμε τον CBJ.
Για να εφαρμόσουμε τον αλγόριθμο FC-CBJ ο οποίος μας ζητείται καλούμε τον cbj_search θέτοντας ως inference συνάρτηση την forward_checking. Δηλαδή όπως ακριβώς καλούσαμε και τον backtracking_search έτσι ώστε να εφαρμόσουμε τον FC.
Τέλος, ομοίως με τις προηγούμενες μεθόδους χρησιμοποιούμε dom/wdeg και lcv για variable-ordering και το value-ordering αντίστοιχα. Επομένως η κλήση του αλγορίθμου γίνεται ως εξής:

**cbj_search(rlfsProblem, csp.dom_wedg, csp.lcv, csp.forward_checking)**

* Περισσότερες λεπτομέρειες σχετικά με την υλοποίηση εξηγούνται αναλυτικά στα σχόλια του κώδικα

## Min conflicts

Η υλοποίηση είναι ίδια με αυτή του aima καθώς δεν χρειάζεται κάποια επιπλέον αλλαγή. Κάθε φορά την καλούμε δίνοντας σαν όρισμα το πρόβλημα που θέλουμε να λύσουμε και τον μέγιστο αριθμό βημάτων που θέλουμε να κάνει (ή παραλείπουμε το δεύτερο όρισμα αφήνοντας τον default αριθμό βημάτων).

### Εκτύπωση αποτελεσμάτων

Έπειτα από την ολοκλήρωση μιας μεθόδου εκτυπώνονται τα εξής:
- Η λύση του προβλήματος. Αν το πρόβλημα δεν έχει λύση εκτυπώνεται None.

Για τις μεθόδους FC, MAC και FC-CBJ εκτυπώνονται επιπλέον:
- Το πλήθος κόμβων που επισκέφτηκε ο αλγόριθμος (Nodes expanded). Η πληροφορία αυτή αποθηκεύεται στην μεταβλητή nassigns η οποία είναι χαρακτηριστικό των αντικειμένων της κλάσης CSP και αυξάνεται μέσω της συνάρτησης assign (υπάρχει ήδη στην υλοποίηση του aima).
- Το πλήθος ελέγχων συνέπειας που πραγματοποιήθηκαν (Number of checks). Η πληροφορία αυτή αποθηκεύεται στο πεδίο checksCount των αντικειμένων CSP και εξηγήθηκε παραπάνω το πώς υπολογίζεται.

Τα Nodes expanded και Number of checks ως μετρικές για την σύγκριση των αλγορίθμων παρακάτω.

# Αποτελέσματα

Τα αποτελέσματα των αλγορίθμων FC, MAC, FC-CBJ για τα instances που μας δόθηκαν είναι τα εξής:

| | FC | | | MAC | | | FC-CBJ | | |
|---|---|---|---|---|---|---|---|---|---|
| **Instances** | Nodes expanded | Number of checks | Execution Time | Nodes expanded | Number of checks | Execution Time | Nodes expanded | Number of checks | Execution Time |
| **11** | 6316 | 1203335 | 0m7.980s | 3003 | 6370124 | 0m16.624s | 6316 | 1203335 | 0m8.683s |
| **2-f24** | 937 | 100754 | 0m0.454s | 200 | 158995 | 0m0.371s | 265 | 21577 | 0m0.238s |
| **2-f25** | 135607 | 22428615 | 1m0.427s | 31002 | 63308219 | 2m47.964s | 3525 | 555494 | 0m1.682s |
| **3-f10** | 68841 | 5904239 | 0m17.431s | 794 | 1144201 | 0m2.431s | 25262 | 3320694 | 0m9.426s |
| **3-f11** | 563197 | 95850540 | 4m1.542s | 22864 | 107552306 | 3m55.535s | 378954 | 59137092 | 2m35.642s |
| **8-f10** | - | - | >20min | 20174 | 35979186 | 1m58.336s | 141041 | 14780817 | 1m32.015s |
| **8-f11** | 506070 | 61216543 | 4m33.983s | 7562 | 18777042 | 0m48.992s | 135444 | 27170626 | 1m32.701s |
| **14-f27** | - | - | >20min | 8068 | 2484951 | 0m26.806s | 30728 | 1661822 | 0m31.471s |
| **14-f28** | 103387 | 5805381 | 1m49.186s | 54773 | 38059144 | 3m16.702s | 5563 | 337953 | 0m5.919s |
| **6-w2** | 683 | 72182 | 0m0.312s | 42 | 397482 | 0m0.529s | 642 | 72220 | 0m0.249s |
| **7-w1-f4** | 46395 | 1721153 | 0m6.907s | 480 | 342293 | 0m0.599s | 10745 | 427492 | 0m2.715s |
| **7-w1-f5** | - | - | >20min | 17923 | 44119585 | 0m53.508s | 3270 | 204655 | 0m0.997s |

## Παρατηρήσεις

- Τα instances με κόκκινο χρώμα είναι unsatisfied σύμφωνα με τα αποτελέσματα των αλγορίθμων τα οποία ταυτίζονται με αυτά που αναφέρθηκαν στο piazza.
- Όπως έχει αναφερθεί και παραπάνω, και οι τρεις μέθοδοι χρησιμοποιούν τον DOM/WDEG για variable-ordering και τον lcv για value-ordering. Επίσης ο MAC χρησιμοποιεί τον αλγόριθμο AC3 (και όχι τον AC3b του aima).
- Οι μέθοδοι FC,MAC και FC-CBJ δεν περιέχουν κάποιον παράγοντα τυχαιότητας (όπως αναφέρθηκε και στην παράγραφο του DOM/WDEG) λόγω της ευρετικής συνάρτησης που όλοι χρησιμοποιούν. Επομένως δεν χρειάζονται πολλαπλές εκτελέσεις για κάθε πρόβλημα καθώς σε κάθε εκτέλεση τα αποτελέσματα (Visited Nodes και Number of checks) είναι πάντα ίδια. Προφανώς από εκτέλεση σε εκτέλεση ο χρόνος εκτέλεσης ίσως έχει κάποια πάρα πολύ μικρή απόκλιση.
- Οι χρόνοι εκτέλεσης προφανώς είναι σχετικοί και διαφέρουν από μηχάνημα σε μηχάνημα. Επίσης σε ορισμένα instances παρατήρησα διαφορετικούς χρόνους εκτέλεσης ανάλογα με την έκδοση της python, κυρίως για τον MAC καθώς με python3.6 αργούσε πολύ και είχε τεράστια διαφορά σε σχέση με την python3.8. Οι εκτελέσεις που φαίνονται στον πίνακα έγιναν με **python3.8.0**
- Τα μεγέθη Nodes expanded και Number of checks εξηγούνται στην αμέσως παραπάνω παράγραφο.
- Ο FC για τα instances στα οποία υπάρχει το ( - ), δεν έβγαζε αποτέλεσμα αφήνοντας τον να τρέχει όση ώρα φαίνεται σε κάθε περίπτωση, έως ότου αναγκάστηκα να τον σταματήσω.
- Οι χρόνοι μετρήθηκαν μέσω της εντολής time κατά την εκτέλεση του προγράμματος (π.χ. time python main.py …)

## Σύγκριση αλγορίθμων

Αρχικά με βάση τους χρόνους εκτέλεσης παρατηρούμε ότι ο FC είναι ο πιο αργός από τους τρεις. Στα περισσότερα instances αργεί σημαντικά σε σχέση με τους άλλους δύο, ενώ για τα δυσκολότερα προβλήματα δεν βρίσκει λύση ακόμα και μετά από πολύ μεγάλο χρονικό διάστημα.
Αντίθετα, οι MAC και FC-CBJ βρίσκουν λύση σε όλα τα προβλήματα και μάλιστα σε αρκετά ικανοποιητικούς χρόνους έως και πολύ καλούς. Παρατηρούμε ότι σε κάποια instances είναι ταχύτερος ο MAC και σε κάποια άλλα είναι ταχύτερος ο FC-CBJ. Για παράδειγμα, για το instance 2-f25 ο MAC χρειάζεται 2 λεπτά και 48 δευτερόλεπτα για να διαπιστώσει ότι δεν έχει λύση, ενώ ο FC-CBJ το διαπιστώνει σε λιγότερο από 2 δευτερόλεπτα. Αντίθετα, για το instance 8-f11 ο MAC διαπιστώνει σε 49 δευτερόλεπτα ότι δεν έχει λύση, ενώ ο FC-CBJ χρειάζεται 1 λεπτό και 32 δευτερόλεπτα.

Όμως, όπως αναφέρθηκε και προηγουμένως, ο χρόνος εκτέλεσης δεν είναι και η καταλληλότερη μετρική. για να συγκρίνουμε αλγορίθμους, συμβάλουν και άλλοι παράγοντες σε αυτόν (π.χ. το hardware του μηχανήματος ή ο φόρτος του μηχανήματος την κάθε χρονική στιγμή). Οπότε χρησιμοποιούμε και τις μετρικές Nodes expanded και Number of checks η οποίες έχουν standard τιμές και με βάση την θεωρία μπορούμε να συγκρίνουμε τους αλγορίθμους και αν ελέγξουμε αν τα αποτελέσματα μας συμβαδίζουν με όσα υποστηρίζονται από την θεωρία.

Όπως γνωρίζουμε από την θεωρία, όσον αφορά τους κόμβους που επισκεπτόμαστε (Expanded Nodes), στα αποτελέσματα μας πρέπει να συναντήσουμε τα:
- Ο FC-CBJ πρέπει να έχει λιγότερους ή το πολύ ίσους Expanded Nodes από ότι ο απλός FC
- Ο MAC πρέπει να έχει λιγότερους ή το πολύ ίσους Expanded Nodes από ότι ο απλός FC

Συγκρίνοντας τα Expanded Nodes των FC-CBJ και FC για κάθε ένα instance παρατηρούμε ότι η πρόταση του πρώτου bullet ισχύει για όλα τα instances.
Συγκρίνοντας τα Expanded Nodes των MAC και FC για κάθε ένα instance παρατηρούμε ότι η πρόταση του δεύτερου bullet επίσης ισχύει για όλα τα instances.
Επίσης παρατηρούμε ότι για τα περισσότερα instances (ακόμα και για instances όπου ο FC-CBJ είναι γρηγορότερος), ο FC-CBJ κάνει expand περισσότερους κόμβους από ότι ο MAC.

Όσον αφορά το πλήθος ελέγχων συνέπειας που γίνονται από κάθε έναν αλγόριθμο, από την θεωρία γνωρίζουμε ότι πρέπει να ισχύει:
- Το πλήθος ελέγχων που πραγματοποιούνται από τον FC-CBJ πρέπει να είναι μικρότερο ή ίσο από του FC

Από τα αποτελέσματα παρατηρούμε ότι αυτό ισχύει για όλα τα instances με εξαίρεση το 6-w2 (δηλαδή ένα πολύ εύκολο πρόβλημα) οπού ο FC κάνει ελάχιστους λιγότερους ελέγχους.

Άρα με βάση τα αποτελέσματα μας και σε συνδυασμό και με την θεωρία συμπεραίνουμε ότι οι αλγόριθμοι έχουν υλοποιηθεί σωστά και τα αποτελέσματα επιβεβαιώνουν την θεωρία.
Επίσης συμπεραίνουμε ότι ο FC είναι ο “χειρότερος” από τους 3 αλγορίθμους.
Οι αλγόριθμοι MAC και FC-CBJ πετυχαίνουν πάρα πολύ καλούς χρόνους ακόμα και για τα δύσκολα προβλήματα.

## Αποτελέσματα Min-Conflicts

Δοκιμάζοντας τον αλγόριθμο min-conflicts για διαφορετικά όρια βημάτων (max steps) και κάνοντας αρκετές εκτελέσεις (καθώς περιέχει και παράγοντα τυχαιότητας όσον αφορά την σειρά επιλογής των μεταβλητών) παρατηρώ ότι δεν βγάζει λύση για κανένα από τα instances που μας δόθηκαν. Δηλαδή είτε φτάνει το μέγιστο όριο βημάτων που έχουμε ορίσει και έτσι τερματίζεται, είτε τρέχει για πολύ ώρα έως ότου το σταματήσω χειροκίνητα (για πολύ μεγάλο όριο βημάτων).
Παρατηρούμε ότι δεν λύνει ούτε τα εύκολα προβλήματα τα οποία οι άλλοι τρεις άλλοι αλγόριθμοι λύνουν σχεδόν ακαριαία.
Επομένως συμπεραίνουμε ότι ο αλγόριθμος min-conflicts δεν είναι καθόλου αποδοτικός για τα συγκεκριμένα προβλήματα.

## Επιπλέον Μελέτη

Δοκιμάζοντας τρόπους να κάνω ακόμα γρηγορότερη την επίλυση των προβλημάτων δοκίμασα μια μικρή παραλλαγή (με όνομα dom_wdeg_extra στο csp.py) της απλής dom/wdeg που χρησιμοποίησα για τα παραπάνω αποτελέσματα την κύριας μελέτης τη άσκησης.
Πιο συγκεκριμένα, όπως εξηγήθηκε και στην παράγραφο dom/wdeg η “παραλλαγμένη” συνάρτηση, όταν δεν μπορεί να επιλέξει κάποια μεταβλητή μέσω του πηλίκου dom/wdeg χρησιμοποιεί την mrv.
Παρατήρησα ότι σε ορισμένα προβλήματα παίρνουμε καλύτερα αποτελέσματα χρησιμοποιώντας την ενώ σε κάποια άλλα χειρότερα. Η ευρετική αυτή ήταν αποδοτικότερη κυρίως για τον FC και τον FC-CBJ ενώ για τον MAC δεν ήταν τόσο αποδοτική καθώς αργούσε πολύ για κάποια προβλήματα τα οποία χρησιμοποιώντας την απλή dom_wdeg λύνονταν γρήγορα.
Επίσης επειδή η dom_wdeg_extra χρησιμοποιεί τον mrv, περιέχει έναν παράγοντα τυχαιότητας. Έτσι είναι αρκετά ασταθής καθώς σε άλλες εκτελέσεις βγάζει καλά αποτελέσματα ενώ σε άλλες εκτελέσεις του ίδιου προβλήματος αργεί πολύ. Αυτός είναι και ένας λόγος που δεν την χρησιμοποίησα για την κύρια μελέτη μου.

Παρακάτω παραθέτω τα αποτελέσματα της επιπλέον μελέτης μου χρησιμοποιώντας την dom_wdeg_extra.
Επειδή υπάρχει ο παράγοντας τυχαιότητας, σε κάθε instance υπάρχουν 5 εκτελέσεις και υπολογίζεται από αυτές ένας μέσος όρος.
Στα παρακάτω αποτελέσματα απουσιάζει ο αλγόριθμος MAC καθώς για αυτόν δεν υπήρχε σημαντική βελτίωση σε σχέση με την κύρια μελέτη με την απλή dom_wdeg. Αντιθέτως για κάποια instances παρουσίαζε τεράστια

| | FC | | | FC-CBJ | | |
|---|---|---|---|---|---|---|
| **Instances** | Nodes expanded | Number of checks | Execution Time (secs) | Nodes expanded | Number of checks | Execution Time (secs) |
| **11** | 6315 | 1203273 | 6.179 | 6315 | 1203266 | 8.71 |
| | 6315 | 1203266 | 7.304 | 6315 | 1203266 | 12.149 |
| | 6315 | 1203266 | 6.617 | 6315 | 1203273 | 14.396 |
| | 6315 | 1203266 | 6.651 | 6315 | 1203273 | 12.057 |
| | 6315 | 1203266 | 6.835 | 6315 | 1203266 | 8.59 |
| **Average** | **6315** | **1203267.4** | **6.7172** | **6315** | **1203268.8** | **11.1804** |
| **2-f24** | 487 | 26722 | 0.216 | 252 | 18944 | 0.112 |
| | 488 | 26792 | 0.17 | 261 | 19466 | 0.169 |
| | 487 | 26735 | 0.165 | 290 | 25582 | 0.131 |
| | 487 | 26761 | 0.169 | 252 | 18729 | 0.191 |
| | 479 | 26000 | 0.166 | 260 | 19473 | 0.109 |
| **Average** | **485.6** | **26602** | **0.1772** | **263** | **20438.8** | **0.1424** |
| **2-f25** | 105157 | 17359245 | 46.341 | 3444 | 548970 | 1.651 |
| | 177230 | 29474369 | 76.498 | 3634 | 578921 | 1.747 |
| | 177230 | 29476627 | 78.944 | 3522 | 552033 | 2.137 |
| | 57022 | 9045958 | 23.213 | 3444 | 548604 | 1.642 |
| | 174437 | 28997830 | 77.247 | 3437 | 548384 | 1.75 |
| **Average** | **138215.2** | **22870805.8** | **60.4486** | **3496.2** | **555382.4** | **1.7854** |
| **3-f10** | 92448 | 9984123 | 26.222 | 25244 | 3326226 | 9.488 |
| | 90259 | 9590290 | 27.532 | 25244 | 3326231 | 10.055 |
| | 90259 | 9590290 | 25.761 | 25244 | 3326226 | 11.163 |
| | 68841 | 5904239 | 17.431 | 25244 | 3326226 | 10.147 |
| | 92448 | 9984123 | 28.047 | 25244 | 3326231 | 11.065 |
| **Average** | **86851** | **9010613** | **24.9986** | **25244** | **3326228** | **10.3836** |
| **3-f11** | 563197 | 95850540 | 241.542 | 378954 | 59146211 | 162.905 |
| | 1842447 | 234098756 | 564.397 | 378954 | 59146211 | 18.755 |
| | 1842447 | 234098756 | 587.363 | 378954 | 59146211 | 156.117 |
| | 563197 | 95850540 | 239.679 | 378954 | 59146211 | 163.823 |
| | 1868820 | 238730404 | 583.323 | 378954 | 59146211 | 157.77 |
| **Average** | **1336021.6** | **179725799.2** | **443.2608** | **378954** | **59146211** | **131.874** |
| **8-f10** | - | - | >20 min | 48748 | 6602769 | 33.745 |
| | - | - | | 140889 | 14780685 | 96.592 |
| | - | - | | 562904 | 47805369 | 290.79 |
| | - | - | | 48748 | 6602769 | 33.745 |
| | - | - | | 140889 | 14778231 | 93.827 |
| **Average** | - | - | | **188435.6** | **18113964.6** | **109.7398** |

| | FC | | | FC-CBJ | | |
|---|---|---|---|---|---|---|
| **Instances** | Nodes expanded | Number of checks | Execution Time (secs) | Nodes expanded | Number of checks | Execution Time (secs) |
| **8-f11** | 204543 | 43435149 | 91.821 | 135428 | 27163980 | 87.781 |
| | 1300371 | 136116991 | 505.754 | 135430 | 27182277 | 96.150 |
| | 303712 | 56504378 | 148.476 | 11259 | 2397380 | 13.168 |
| | 506070 | 61216543 | 273.983s | 135428 | 27163980 | 101.332 |
| | 204543 | 43435149 | 93.944 | 11259 | 2397380 | 14.257 |
| **Average** | **503847.8** | **68141642** | **209.99875** | **85760.8** | **17260999.4** | **62.5376** |
| **14-f27** | - | - | >20 min | 24983 | 1315895 | 27.731 |
| | - | - | | 84647 | 4449682 | 92.094 |
| | - | - | | 24983 | 1315895 | 27.945 |
| | - | - | | 24983 | 1315895 | 28.095 |
| | - | - | | 24983 | 1315895 | 26.324 |
| **Average** | - | - | | 36915.8 | 1942652.4 | 40.4378 |
| **14-f28** | 51618 | 2665621 | 52.823 | 3728 | 222429 | 4.3087 |
| | 105200 | 6038196 | 99.119 | 3728 | 222429 | 4.254 |
| | 105200 | 6038196 | 104.221 | 5555 | 337093 | 5.6023 |
| | 51618 | 2665621 | 51.356 | 3728 | 222429 | 4.086 |
| | 105200 | 6038196 | 98.281 | 3728 | 222429 | 4.235 |
| **Average** | **83767.2** | **4689166** | **81.16** | **4093.4** | **245361.8** | **4.4972** |
| **6-w2** | 683 | 72182 | 0.312 | 297 | 50606 | 0.111 |
| | 62650 | 5899275 | 15.854 | 17171 | 1456227 | 3.919 |
| | 7302 | 1408654 | 2.653 | 297 | 50606 | 0.119 |
| | 7302 | 1408654 | 2.820 | 17171 | 1456227 | 4.145 |
| | 7302 | 1408654 | 2.104 | 297 | 50606 | 0.114 |
| **Average** | **17047.8** | **2039483.8** | **4.7486** | **7046.6** | **612854.4** | **1.6816** |
| **7-w1-f4** | 46395 | 1721153 | 6.907 | 10421 | 410197 | 2.414 |
| | 46417 | 1721835 | 6.315 | 10745 | 427492 | 3.033 |
| | 141543 | 4726143 | 20.180 | 74131 | 5839872 | 31.701 |
| | 46395 | 1721153 | 7.377 | 10745 | 427492 | 2.869 |
| | 46417 | 1721835 | 6.315 | 10767 | 428174 | 2.850 |
| **Average** | **65433.4** | **2322423.8** | **9.4188** | **23361.8** | **1506645.4** | **8.5734** |
| **7-w1-f5** | - | - | >20 min | 3269 | 204482 | 0.967 |
| | - | - | | 3269 | 204482 | 0.939 |
| | - | - | | 3269 | 204482 | 0.917 |
| | - | - | | 3269 | 204521 | 0.904 |
| | - | - | | 3292 | 205192 | 0.907 |
| **Average** | - | - | | **3273.6** | **204631.8** | **0.9268** |

## Πρόβλημα 2

Αρχικά πρέπει να μοντελοποιήσουμε το πρόβλημα μας σαν πρόβλημα ικανοποίησης περιορισμών.
Για να ορίσουμε το πρόβλημα πρέπει να ορίσουμε: τις μεταβλητές (variables), το πεδίο τιμών της κάθε μεταβλητής (domain) και ένα σύνολο από περιορισμούς.
Για να γίνει ευκολότερα αντιληπτό το πρόβλημα σκεφτόμαστε το δωμάτιο σαν ένα σύστημα αξόνων x και y όπου x είναι το πλάτος και y είναι το μήκος. Επίσης, θεωρούμε ότι η αρχή των αξόνων είναι η κάτω αριστερά γωνία του δωματίου (όπως αυτή φαίνεται στο σχήμα της εκφώνησης) και τέλος ότι το πλάτος παίρνει τιμές από το 0 μέχρι το 300 και το μήκος από το 0 μέχρι το 400 (δηλαδή τα όρια του δωματίου).
Επίσης υποθέτουμε (όπως υποδείχθηκε και στο piazza) ότι τα έπιπλα δεν μπορούν να περιστραφούν
Παρατήρηση: Στην εκφώνηση δεν διαχωρίζεται ποιό από τα δύο είναι το πλάτος και ποιό το μήκος όποτε ως θεωρώ πλάτος τον παράλληλο τοίχο και ως μήκος τον κάθετο. (Φαίνεται και στο σχήμα της επόμενης σελίδας)

Μεταβλητές:
Στο πρόβλημα μας οι μεταβλητές είναι τα έπιπλα που θέλουμε να τοποθετήσουμε στο δωμάτιο, δηλαδή το κρεβάτι, το γραφείο, η καρέκλα και ο καναπές. Το κάθε ένα έπιπλο έχει προκαθορισμένο πλάτος, μήκος (ή βάθος (αντιμετωπίζουμε το βάθος σαν μήκος)) και ύψος. Θα θέλαμε να κρατάμε τα δεδομένα αυτά για κάθε έπιπλο οπότε μπορούμε να ορίσουμε σαν μεταβλητή το σύνολο αυτών των πληροφοριών. Πιο συγκεκριμένα ορίζουμε την κάθε μεταβλητή σαν το σύνολο (π.χ. tuple) (έπιπλο, πλάτος, μήκος, ύψος). Οι πληροφορίες αυτές παραμένουν σταθερές.
Άρα οι μεταβλητές του προβλήματος μας είναι: (κρεβάτι,100,200,80), (γραφείο,160,80,90), (καρέκλα,41,44,57) και (καναπές,221,103,84).

**Παρατήρηση:**
Στο πρόβλημα μας θα "αγνοήσουμε" το ύψος στις αναθέσεις τιμών και στους περιορισμούς, καθώς τα έπιπλα δεν μπορούν να είναι το ένα πάνω στο άλλο και ούτε μπορούν να αιωρούνται. Δηλαδή αυτό σημαίνει ότι όλα τα έπιπλα θα πρέπει να "πατάνε στο έδαφος", δηλαδή όλα χωράνε στο δωμάτιο όσον αφορά το ύψος. Άρα το ύψος δεν επηρεάζει κάπως το πρόβλημα μας για αυτό και δεν το λαμβάνω υπόψιν, έτσι ώστε να απλοποιηθεί το πρόβλημα μας.

Domain:
Στο πρόβλημα μας, ανάθεση τιμής σε μια μεταβλητή ουσιαστικά σημαίνει να αναθέσουμε μια θέση του δωματίου στο έπιπλο. Επειδή γνωρίζουμε τις διαστάσεις την κάθε μεταβλητής (επίπλου) αρκεί να αναθέσουμε μία τιμή στην μία γωνία του επίπλου. Πιο συγκεκριμένα χρειάζεται να αναθέσουμε ένα σετ (πλάτος, μήκος) το οποίο θα αντιστοιχεί στην μία γωνία του επίπλου. Εφόσον γνωρίζουμε τις διαστάσης του κάθε επίπλου μπορούμε να βρούμε και τις συντεταγμένες των υπόλοιπων γωνιών του επίπλου.
Επιλέγω η τιμή που θα ανατίθεται κάθε φορά, να αντιστοιχεί στην κάτω αριστερά γωνία του επίπλου (υποθέτοντας ότι τα έπιπλα αναπαρίστανται ως ορθογώνια παραλληλόγραμμα).
Οπότε με λίγα λόγια κάθε φορά που αναθέτουμε τιμή σε μία μεταβλητή, αναθέτουμε συντεταγμένες στην κάτω αριστερά γωνία του επίπλου.
Όσον αφορά τις τιμές που μπορούν να ανατεθούν σε κάθε μεταβλητή, η μόνη προϋπόθεση είναι τα έπιπλα να μην υπερβαίνουν τα όρια του δωματίου.
Γνωρίζουμε τα "όρια" του δωματίου τα οποία είναι 0<=πλάτος<=300 και 0<=μήκος<=400.
Επίσης, γνωρίζουμε τις διαστάσεις της κάθε μεταβλητής, δηλαδή το μήκος της και το πλάτος της.
Τις τιμές που αναθέτουμε στις μεταβλητές θα τις ορίσω σαν π (για το πλάτος) και μ (για το μήκος). Δηλαδή σε μια μεταβλητή αναθέτουμε ένα σετ (π,μ).
Για να είναι ένα έπιπλο στα όρια του δωματίου πρέπει να ισχύει 0<=μ<=(400-μήκοςΜεταβλητής) και 0<=π<=(300-πλάτοςΜεταβλητής) αφού όπως είπαμε αναθέτουμε τις "συντεταγμένες" στην κάτω αριστερά γωνία.
Άρα τα domain των μεταβλητών μας είναι:

- Για το κρεβάτι:

domain[ (κρεβάτι,100,200,80) ] = (π,μ) όπου $0 \leq \pi \leq 200$ και $0 \leq \mu \leq 200$

- Για το γραφείο:

domain[ (γραφείο,160,80,90) ] = (π,μ) όπου $0 \leq \pi \leq 140$ και $0 \leq \mu \leq 320$

- Για την καρέκλα:

domain[ (καρέκλα,41,44,57) ] = (π,μ) όπου $0 \leq \pi \leq 259$ και $0 \leq \mu \leq 356$

- Για τον καναπέ:

domain[ (καναπές,221,103,84) ] = (π,μ) όπου $0 \leq \pi \leq 79$ και $0 \leq \mu \leq 297$

Περιορισμοί:
Οι περιορισμοί που έχουμε είναι:

1. Τα έπιπλα δεν πρέπει να μπλοκάρουν την πόρτα. (θα μπορούσε να ληφθεί υπόψιν κατά την κατασκευή των domains αλλά επέλεξα να το προσθέσω σαν constrain καθώς θα περιέπλεκε αρκετά την κατασκευή των domains).
2. Τα έπιπλα δεν πρέπει να εφάπτονται ή να πατάνε το ένα πάνω στο άλλο.
3. Το γραφείο θα πρέπει να είναι δίπλα σε κάποια είσοδο φωτός στο δωμάτιο.

Περιορισμός 1:
Η πόρτα έχει πλάτος 100 εκατοστά. Δηλαδή για να ανοίγει χρειάζεται έναν χώρο 100x100 (στην κάτω αριστερά γωνία του σχήματος της εκφώνησης). Αυτό σημαίνει ότι για μία ανάθεση πρέπει να τηρείται ο περιορισμός: $\pi > 100$ ή $\mu > 100$ . Δηλαδή κανένα έπιπλο να μην βρίσκεται στο τετράγωνο 100x100 της κάτω αριστερά γωνίας.

Περιορισμός 2:
Για να ελέγξουμε αν δύο έπιπλα εφάπτονται ή πατάνε το ένα πάνω στο άλλο πρέπει να βρούμε και τις συντεταγμένες της πάνω δεξιάς γωνίας. Μπορούμε να το υπολογίσουμε, καθώς έχουμε ήδη τις συντεταγμένες της κάτω αριστερά γωνίας (από την ανάθεση) και τις διαστάσεις (πλάτος και μήκος) της κάθε μεταβλητής.
Θα συμβολίζω τις συντεταγμένες της πάνω δεξιά γωνίας ως (π2,μ2) όπου π2=π+πλάτοςΜεταβλητής (όπου π είναι η τιμή της ανάθεσης και πλάτος το σταθερό πλάτος του επίπλου) και μ2=μ+μήκοςΜεταβλητής.
Κατά τον έλεγχο του περιορισμού συγκρίνουμε κάθε φορά 2 έπιπλα (μεταβλητές) έστω έπιπλο1 και έπιπλο2.
Υπολογίζουμε τις συντεταγμένες των πάνω δεξιά γωνιών του κάθε επίπλου με τον τρόπο που αναφέρθηκε παραπάνω. Άρα για το έπιπλο1 έχουμε: (έπιπλο1.π2,έπιπλο1.μ2) και για το έπιπλο2 έχουμε: (έπιπλο2.π2,έπιπλο2.μ2).
Υπάρχουν οι εξής 4 περιπτώσεις όπου 2 έπιπλα δεν εφάπτονται, ούτε είναι το ένα πάνω στο άλλο:
- Αν το ένα έπιπλο “ξεκινάει” (έστω το έπιπλο2) από μεγαλύτερο ύψος από ότι τελειώνει το άλλο έπιπλο (έπιπλο1). Δηλαδή υποθέτοντας ότι βλέπουμε το δωμάτιο από ψηλά (όπως στην εικόνα της εκφώνησης), το ένα έπιπλο βρίσκεται πιο “πάνω” από το άλλο. Στην περίπτωση αυτή ισχύει ότι το μήκος της πάνω δεξιά γωνίας του πρώτου έπιπλου είναι μικρότερο από ότι το μήκος της κάτω αριστερά γωνιάς του δεύτερου επίπλου. Δηλαδή : **έπιπλο1.μ2<έπιπλο2.μ**
- Το ακριβώς αντίστροφο από το πρώτο bullet, δηλαδή το έπιπλο2 να βρίσκεται “κάτω” από το έπιπλο1. Στην περίπτωση αυτή ισχύει ότι το μήκος της πάνω δεξιάς γωνίας του έπιπλου2 είναι μικρότερο από ότι το μήκος της κάτω αριστερά γωνίας του πρώτου επίπλου. Δηλαδή: **έπιπλο1.μ>έπιπλο2.μ2**.
- Αν το πρώτο έπιπλο βρίσκεται πιο “αριστερά” από το δεύτερο, δηλαδή το πλάτος της πάνω δεξιάς γωνίας (δηλαδή το πλάτος στο οποίο “τελειώνει” το έπιπλο1) είναι μικρότερο από ότι το πλάτος της κάτω αριστερά γωνίας του δεύτερου επίπλου (δηλαδή το πλάτος στο οποίο “αρχίζει” το έπιπλο2). Στην περίπτωση αυτή ισχύει: **έπιπλο1.π2<έπιπλο2.π**.
- Το αντίθετο από το bullet3 δηλαδή αν το έπιπλο1 βρίσκεται δεξιά από το έπιπλο2. Στην περίπτωση αυτή ισχύει ότι το πλάτος της πάνω δεξιά γωνίας του επίπλου2 είναι μικρότερο από το πλάτος της κάτω αριστερά γωνίας του επίπλου1. Δηλαδή ισχύει: **έπιπλο1.π>έπιπλο2.π2**.

Προφανώς μπορεί να ισχύουν και παραπάνω από μία συνθήκες ταυτόχρονα, αν π.χ. τα έπιπλα βρίσκονται σε διαγώνια θέση μεταξύ τους.
Άρα, συνολικά, για να τηρείται ο περιορισμός 2 για για δύο μεταβλητές πρέπει να ισχύει:

**έπιπλο1.μ2<έπιπλο2.μ** ή **έπιπλο1.μ>έπιπλο2.μ2** ή **έπιπλο1.π2<έπιπλο2.π** ή **έπιπλο1.π>έπιπλο2.π2**

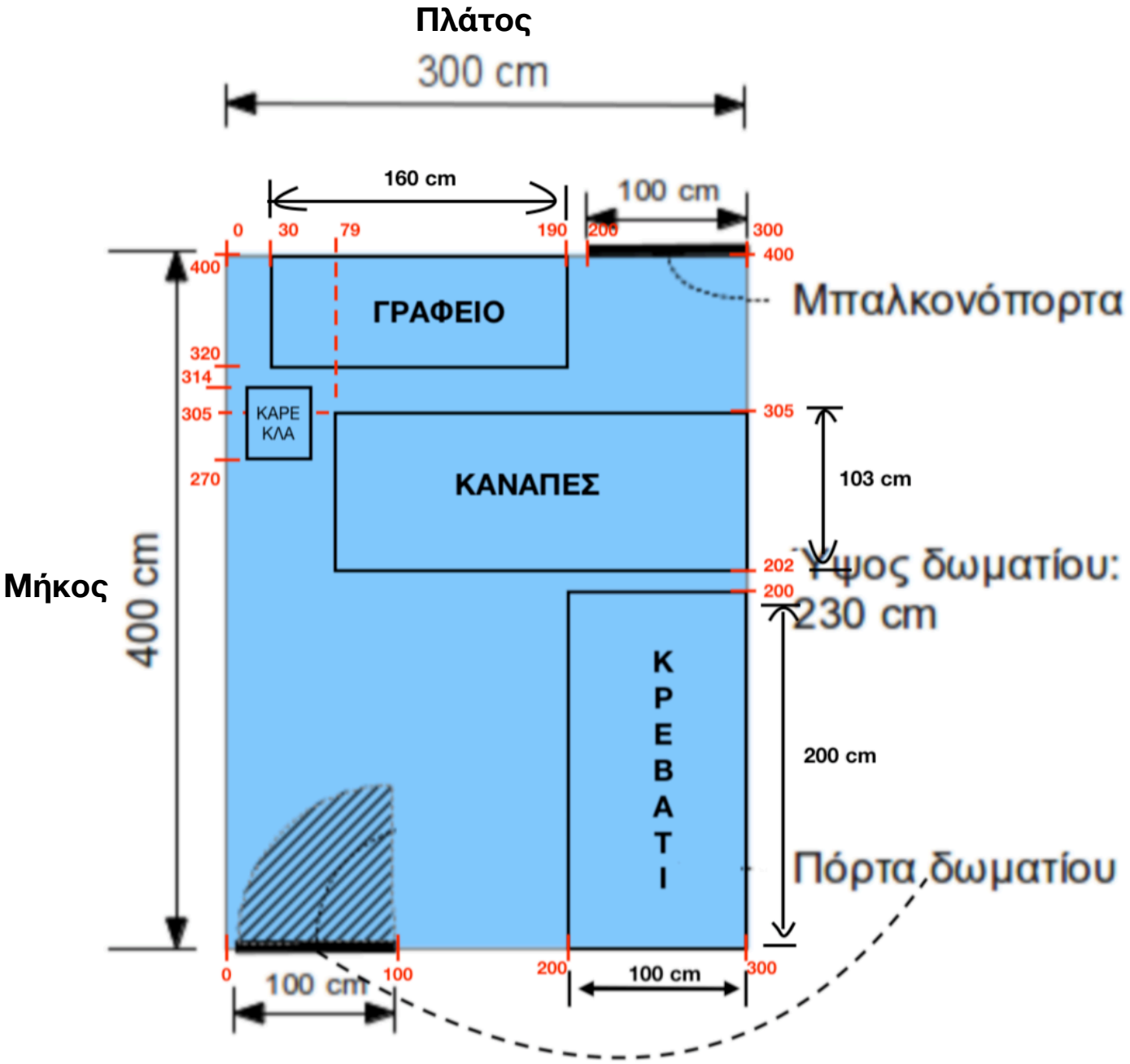
Περιορισμός 3:
Έστω ότι η μέγιστη απόσταση που μπορεί να έχει το γραφείο από το παράθυρο είναι 15 εκατοστά. Επίσης θεωρούμε σαν πηγή φωτός την μπαλκονόπορτα.
Αρχικά έχουμε ως δεδομένο τις συνταγμένες των δύο άκρων της μπαλκονόπορτας οι οπόιες είναι (πλάτος=200,μήκος=400) (αριστερό άκρο της μπαλκονόπορτας) και (300,400) (δεξί άκρο μπαλκονόπορτας).
Επίσης υπολογίζουμε την πάνω δεξιά γωνία του γραφείου και την πάνω αριστερά γωνία του γραφείου με τον τρόπο ο οποίος περιγράφηκε στον περιορισμό 2.
Τέλος, υπολογίζουμε την ευκλείδεια απόσταση μεταξύ των παραπάνω ζευγαριών σημείων, δηλαδή υπολογίζουμε:
- ευκλείδεια_απόσταση( πάνω_δεξιά_γωνία_γραφείου , δεξί_άκρο_μπαλκονόπορτας )
- ευκλείδεια_απόσταση( πάνω_αριστερή_γωνία_γραφείου , δεξί_άκρο_μπαλκονόπορτας )
- ευκλείδεια_απόσταση( πάνω_δεξιά_γωνία_γραφείου , αριστερό_άκρο_μπαλκονόπορτας )
- ευκλείδεια_απόσταση( πάνω_αριστερή_γωνία_γραφείου , αριστερό_άκρο_μπαλκονόπορτας )

Αν μία από τις 4 παραπάνω αποστάσεις είναι μικρότερη από την μέγιστη απόσταση που έχουμε ορίσει (δηλαδή 15 εκατοστά) τότε το γραφείο είναι αρκετά κοντά στην πηγή φωτός το οποίο σημαίνει ότι τηρείται ο περιορισμός.

Συνολικά έχουμε ορίσει τις μεταβλητές, τα domain των μεταβλητών τους περιορισμούς καθώς και την μεθοδολογία ελέγχου των περιορισμών αυτών. Δηλαδή έχουμε ορίσει πλήρως το πρόβλημα ικανοποίησης περιορισμών που προσπαθούμε να λύσουμε.

Το μόνο που απομένει είναι να εντοπίσουμε το κατά πόσο έχει λύση το πρόβλημα αυτό.

Λαμβάνοντας υπόψιν όλα τα παραπάνω και ακολουθώντας την λογική την οποία ακολουθεί ένας αλγόριθμος επίλυσης προβλημάτων ικανοποίησης περιορισμών μπορούμε να βρούμε μια λύση για το πρόβλημα μας η οποία τηρεί όλους τους περιορισμούς αλλά και τις υποθέσεις που κάναμε (όπως ότι τα έπιπλα δεν μπορούν να περιστραφούν και το ότι το γραφείο πρέπει να έχει απόσταση το πολύ 15 εκατοστά από την μπαλκονόπορτα). Η λύση είναι η εξής.




Παρατηρούμε ότι τηρούνται όλοι οι περιορισμοί, άρα το πρόβλημα μας έχει λύση.

Παρατήρηση:
Μπορούμε να τοποθετήσουμε την καρέκλα σε οποιοδήποτε σημείο καθώς δεν υπάρχει κάποιος περιορισμός σχετικά με την απόσταση της από το γραφείο, δεν είναι απαραίτητο να τοποθετηθεί κοντά στο γραφείο όπως την τοποθέτησα εγώ στην παραπάνω εικόνα.

## Πρόβλημα 3

Ερώτημα α

Για να μοντελοποιήσουμε το. πρόβλημα σαν πρόβλημα ικανοποίησης περιορισμών θα ακολουθήσουμε την ίδια διαδικασία με το πρόβλημα 2. Δηλαδή, θα ορίσουμε τις μεταβλητές, το domain της κάθε μεταβλητής και τους περιορισμούς.

Μεταβλητές
Στο πρόβλημα μας οι μεταβλητές είναι οι 5 ενέργειες οι οποίες θέλουμε να χρονοπρογραμματίσουμε. Δηλαδή έχουμε τις 5 μεταβλητές: A1, A2, A3, A4, A5.

Domain
Όλες οι μεταβλητές χρειάζονται τον ίδιο σταθερό χρόνο για να ολοκληρωθούν ο οποίος είναι ίσος με 60. Άρα η μόνη τιμή που χρειάζεται να αναθέσουμε είναι η στιγμή εκκίνησης της κάθε ενέργειας. Η στιγμή ολοκλήρωσης της κάθε ενέργειας είναι κάθε φορά η στιγμή εκκίνησης συν 60 λεπτά.
Όλες οι ενέργειες μπορούν να αρχίσουν στις 9:00, στις 10:00 ή στις 11:00. Άρα όλες οι μεταβλητές έχουν το ίδιο domain και αυτό είναι το σύνολο {9:00, 10:00, 11:00}.

Περιορισμοί:
Όπως προαναφέρθηκε, σε κάθε μεταβλητή αναθέτουμε μια στιγμή εκκίνησης. Άρα σε όσες μεταβλητές έχουν ανατεθεί τιμές, έχουν μια στιγμή εκκίνησης. Στην πληροφορία αυτή θα την συμβολίζω με: Μεταβλητή.Εκκίνηση (π.χ. A1.Εκκίνηση).
Επίσης για λόγους κατανόησης, τα 60 λεπτά που χρειάζεται μια ενέργεια για να ολοκληρωθεί θα τα αναπαριστώ σαν μία ώρα δηλαδή: 1:00.
Έχουμε τους παρακάτω περιορισμούς:
- Η A1 πρέπει να αρχίσει μετά την A3
  Δηλαδή πρέπει να ισχύει: **A1.Εκκίνηση>A3.Εκκίνηση**
- Η A3 πρέπει να αρχίσει πριν την A4 και μετά την A5
  Δηλαδή πρέπει να ισχύει **A3.Εκκίνηση<A4.Εκκίνηση** και **A3.Εκκίνηση>A5.Εκκίνηση**
- Η A2 δεν μπορεί να εκτελείται την ίδια ώρα με την A1 ή την A4

Γενικά για να ελέγξουμε ότι η A2 δεν εκτελείται την ίδια ώρα με την A1 πρέπει να ισχύει ότι:
**( (A2.Εκκίνηση+1:00)<=A1.Εκκίνηση ή (A1.Εκκίνηση+1:00)<=A2.Εκκίνηση).** Ομοίως για την A2 με την A4 πρέπει να ισχύει ότι: **( (A2.Εκκίνηση+1:00)<=A4.Εκκίνηση ή (A4.Εκκίνηση+1:00)<=A2.Εκκίνηση)**. Βέβαια στην περίπτωση μας επειδή οι δυνατές στιγμές εκκίνησης απέχουν μεταξύ τους 1 ώρα και επίσης κάθε ενέργεια χρειάζεται 1 ώρα για να ολοκληρωθεί αρκεί απλά να ελέγξουμε αν έχουν ίδια ώρα εκκίνησης. Δηλαδή για την A2 με την A1 αρκεί να ισχύει: **A2.Εκκίνηση!=A1.Εκκίνηση**
και για την A2 με την A4 αρκεί: **A2.Εκκίνηση!=A4.Εκκίνηση**.
- Η A4 δεν μπορεί να αρχίσει στις 10:00
  Δηλαδή πρέπει να ισχύει **A4.Εκκίνηση!=10:00**.

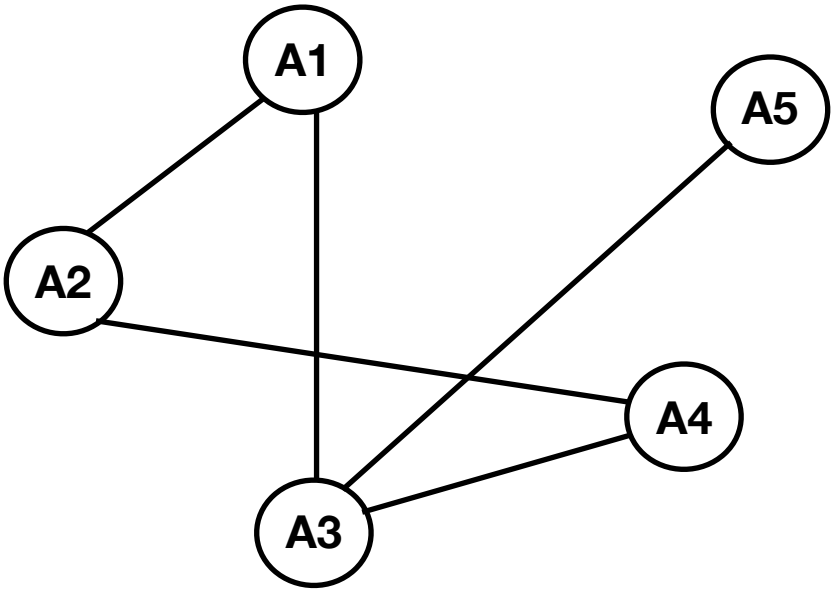
Άρα ορίσαμε όλους τους περιορισμούς τους οποίους πρέπει να ισχύουν καθώς και τις συνθήκες τις οποίες πρέπει να ελέγχουμε για να δούμε εάν τηρείται ο κάθε περιορισμός.

Εφόσον ορίσαμε κατάλληλα τις μεταβλητές τα domain τους και τους περιορισμούς (και την μέθοδο ελέγχου τους), έχουμε μοντελοποιήσει κατάλληλα το πρόβλημα σαν πρόβλημα ικανοποίησης περιορισμών.

Ερώτημα β

Για να σχεδιάσουμε τον γράφο περιορισμών του παραπάνω προβλήματος (όπως και κάθε άλλου προβλήματος), αρχικά αντιστοιχίζουμε τις μεταβλητές με τους κόμβους του γράφου. Επίσης, ένας περιορισμός μεταξύ δύο μεταβλητών αντιστοιχεί σε μια ακμή μεταξύ των αντίστοιχων δύο κόμβων.

Άρα ο γράφος που προκύπτει για το πρόβλημα μας είναι:




Ερώτημα γ

Εφαρμόζουμε μια επανάληψη του αλγορίθμου AC3 έτσι ώστε όλες οι ακμές να γίνουν συνεπείς σύμφωνα με τους περιορισμούς του προβλήματος.
Οι περιορισμοί του προβλήματος (όπως εξηγήθηκαν στο ερώτημα Α) είναι:

- A1 > A3 και αντίστροφα A3<A1
- A2 != A1 , A1 != A2
- A2 != A4 , A4 != A2
- A3 < A4 , A4 > A3
- A3 > A5 , A5 < A3
- A4 != 10:00 , 10:00 != A4

Υποθέτουμε ότι η σειρά που επιλέγουμε τις μεταβλητές για ανάθεση τιμών είναι A1, A2, A3, A4, A5.
Και η σειρά επιλογής τιμών είναι: 9:00, 10:00, 11:00.
Τα αρχικά domains των μεταβλητών είναι:

| Varables | Domains | | |
|---|---|---|---|
| **A1** | 9:00 | 10:00 | 11:00 |
| **A2** | 9:00 | 10:00 | 11:00 |
| **A3** | 9:00 | 10:00 | 11:00 |
| **A4** | 9:00 | 10:00 | 11:00 |
| **A5** | 9:00 | 10:00 | 11:00 |

**Ανάθεση A1 = 9:00**

Επειτά από την ανάθεση ελέγχουμε τους περιορισμούς που έχουν στο δεξιό μέλος τους την μεταβλητή A1. Σύμφωνα με τους περιορισμούς αυτούς, αφαιρούμε όσες τιμές από τα domain των γειτόνων δεν τηρούν τους περιορισμούς αυτούς.

| Varables | Domains | | |
|---|---|---|---|
| **A1** | 9:00 | 10:00 | 11:00 |
| **A2** | ~~9:00~~ | 10:00 | 11:00 |
| **A3** | ~~9:00~~ | 1~~0:00~~ | ~~11:00~~ |
| **A4** | 9:00 | 10:00 | 11:00 |
| **A5** | 9:00 | 10:00 | 11:00 |

Οι γείτονες της A1 είναι: A2 και A3

Περιορισμοί που ελέγχονται:

- A2 != A1
- A3 < A1

Άρα αφαιρούμε τις inconsistent τιμές από τα domains των A2 και A3. Τα νέα domains που προκύπτουν φαίνονται στον διπλανό πίνακα.

* με κόκκινη γραμμή απεικονίζονται οι τιμές που αφαιρέθηκαν, ενώ στο γαλάζιο κουτάκι είναι η τιμή που έχει ανατεθεί στην αντίστοιχη μεταβλητή

Παρατηρούμε ότι έπειτα από την ανάθεση τιμής στην A1, το domain της A3 είναι άδειο. Δηλαδή, έχουμε ασυνέπεια. Επομένως, δοκιμάζουμε την επόμενη τιμή για την A1 ξεκινώντας από τα αρχικά domains.

**Ανάθεση A1 = 10:00**

Ομοίως με πριν ελέγχουμε τους περιορισμούς που έχουν στο δεξιό μέλος τους την μεταβλητή A1 και αρχικά ενημερώνουμε τα domains των γειτόνων της.
Αφού ενημερωθούν τα domains των γειτόνων, για όσα από αυτά άλλαξαν πρέπει να ακολουθήσουμε την ίδια διαδικασία. Δηλαδή να ελέγξουμε όσους περιορισμούς έχουν στο δεξί μέλος την μεταβλητή αυτή και να ενημερώσουμε τα domains των μεταβλητών στο αριστερό μέλος.
Η διαδικασία αυτή ακολουθείται έως ότου έχουμε συνέπεια στις ακμές.

| Varables | Domains | | |
|---|---|---|---|
| **A1** | | 10:00 | 11:00 |
| **A2** | 9:00 | ~~10:00~~ | 11:00 |
| **A3** | 9:00 | ~~10:00~~ | ~~11:00~~ |
| **A4** | ~~9:00~~ | 10:00 | 11:00 |
| **A5** | ~~9:00~~ | ~~10:00~~ | ~~11:00~~ |

Περιορισμοί που ελέγχονται:
- A2 != A1
- A3 < A1

→ Λόγω της ανάθεσης του A1

- A4 != A2 → Λόγω της αλλαγής του domain της A2
- A4 > A3
- A5 < A3

→ Λόγω της αλλαγής του domain της A3

* με πράσινο χρώμα είναι ο περιορισμός ο οποίος είναι συνεπής. Οι περιορισμοί που δεν έχουν πράσινο χρώμα δεν είναι συνεπής και έτσι χρειάζεται να γίνουν αλλαγές στα domains των μεταβλητών στο αριστερό μέλος του περιορισμού. Οι αλλαγές που γίνονται φαίνονται με το αντίστοιχο χρώμα στον πίνακα.

Παρατηρούμε ότι λόγω του περιορισμού A5<A3 το domain της A5 αδειάζει, δηλαδή δεν υπάρχει καμία επιτρεπτή τιμή για την μεταβλητή A5. Έτσι κάνουμε backtrack, επιλέγοντας την επόμενη τιμή προς ανάθεση για την μεταβλητή A1. Ξεκινάμε, πάλι, από τα αρχικά domains.

**Ανάθεση A1 = 11:00**

Ακολουθούμε ακριβώς την ίδια διαδικασία με πριν έως ότου οι ακμές είναι συνεπείς.

| Varables | Domains | | |
|---|---|---|---|
| **A1** | | | 11:00 |
| **A2** | 9:00 | 10:00 | ~~11:00~~ |
| **A3** | ~~9:00~~ | 10:00 | ~~11:00~~ |
| **A4** | ~~9:00~~ | ~~10:00~~ | 11:00 |
| **A5** | 9:00 | ~~10:00~~ | ~~11:00~~ |

Περιορισμοί που ελέγχονται:
- A2 != A1
- A3 < A1

→ Λόγω της ανάθεσης του A1

- A4 != A2 → Λόγω της αλλαγής του domain της A2
- A4 > A3
- A5 < A3

→ Λόγω της αλλαγής του domain της A3

- 10:00 != A4 → Λόγω της αλλαγής του domain της A4
- A2 != A4
- A3 < A4

→ Λόγω της αλλαγής του domain της A4

- A3 > A5 → Λόγω της αλλαγής του domain της A5
- A4 > A3
- A5 < A3

→ Λόγω της αλλαγής του domain της A3

Παρατηρούμε ότι όλες οι ακμές είναι συνεπής. Άρα συνεχίζουμε με ανάθεση τιμής στην επόμενη μεταβλητή, δηλαδή στην A2. Τα domains των μεταβλητών είναι αυτά που προέκυψαν στον τελευταίο πίνακα.

**Ανάθεση A2 = 9:00**

Ακολουθούμε ακριβώς την ίδια διαδικασία με πριν έως ότου οι ακμές είναι συνεπείς.

| Varables | Domains | | |
|---|---|---|---|
| **A1** | | | 11:00 |
| **A2** | 9:00 | 10:00 | |
| **A3** | | 10:00 | |
| **A4** | | | 11:00 |
| **A5** | 9:00 | | |

Περιορισμοί που ελέγχονται:
- A4 != A2 → Λόγω της ανάθεσης του A2

Η μόνη ακμή μεταξύ της μεταβλητής A2 και κάποιας unassigned μεταβλητής είναι η (A4,A2). Παρατηρούμε ότι σύμφωνα με την ανάθεση της τιμής της A2 η ακμή παραμένει συνεπής. Αυτό σημαίνει ότι δεν χρειάζεται να αλλάξει το domain καμίας μεταβλητής, όλες οι ακμές παραμένουν συνεπείς και επομένως ο έλεγχος σταματάει. Άρα μπορούμε να συνεχίσουμε με ανάθεση τιμής στην επόμενη μεταβλητή.

Όμως, παρατηρούμε ότι στα domains των υπολοίπων μεταβλητών περιέχεται μία μόνο τιμή.
Λαμβάνοντας υπόψιν ότι υπάρχει μόνο μια διαθέσιμη τιμή για κάθε μεταβλητή και ότι όλες οι ακμές είναι συνεπής, συμπεραίνουμε ότι έχουμε βρει λύση στο πρόβλημα μας.
Συγκεκριμένα, αναθέτουμε τις αντίστοιχες τιμές τις τρεις υπόλοιπες μεταβλητές και η λύση που προκύπτει είναι:

- A1 = 11:00
- A2 = 9:00
- A3 = 10:00
- A4 = 11:00
- A5 = 9:00