Τεχνητή Νοημοσύνη 1 - Χειμερινό 2020/2021
Εργασία 3

Ροβιθάκης Ιωάννης | sdi1800164

Πρόβλημα 1:

Έχω καλύψει τα ερωτήματα **1.** και **2.** στα αρχεία πηγαίου κώδικα:
main.py // Το κύριο πρόγραμμα, το οποίο διαβάζει τα δεδομένα, αρχικοποιεί ένα πρόβλημα csp, και στη συνέχεια καλεί την επιλεγμένη συνάρτηση για επίλυση του προβλήματος αυτου και τυπώνει τα αποτελέσματα με βάση διαφορα κριτήρια
my_csp.py // Μια ελαφρά τροποποιημένη εκδοχή του csp.py από το github του βιβλίου AIMA για την αναπαράσταση και επίλυση csp με διαφορους τρόπους.

(Μαζί με τα δύο αρχεία αυτά, παραδίδω και τα αρχεία search.py και utils.py, τα οποία χρησιμοποιεί εσωτερικά το my_csp.py και είναι ακριβώς τα αντιστοιχα αρχεία από το github του βιβλίου AIMA, χωρίς καμία αλλαγή)

3. Σχολιασμός κώδικα, παρατηρήσεις και παραδοχές

i) Για να μπορώ να δοκιμάσω εύκολα τις διαφορες περιπτώσεις αλγοριθμων στα διαθέσιμα προβλήματα, έχω ένα σύνολο από " defines" στο main.py, από τα οποία επιλέγω αυτά που επιθυμώ κάθε φορά πριν την εκτέλεση.

ii) Ως κριτήρια σύγκρισης με βάση την θεωρία έχουμε τα εξής:
> Χρονική/χωρική πολυπλοκότητα χείριστης πείπτωσης
> Χρόνος εκτέλεσης
> Πλήθος κόμβων που επισκεπτόμαστε στο δέντρο αναζήτησης
> Πλήθος ελέγχων συνέπειας που πραγματοποιούνται

Για το ερώτημα αυτό χρησιμοποίησα κυρίως το πλήθος ελέγχων περιορισμών και το πλήθος κόμβων ως μέτρα αξιολόγησης μιας μεθόδου, καθώς ο χρόνος όπως είπαμε στο μάθημα μπορεί να διαφέρει από υπολογιστή σε υπολογιστή και απο μικρές αλλαγές -διαφορές στην υλοποίηση, ενώ η πολυπλοκότητα απο μόνη της δεν προσφέρει ιδιαίτερο διαχωρισμό.
> Όλες οι παραπάνω μέθοδοι έχουν εκθετική χρονική πολυπλοκότητα για το worst case
> Το πλήθος των κόμβων μας δείχνει σαν κριτήριο το πόσο πολύ "επεκτείνεται" μια μέθοδος στον "γράφο" της αναζήτησης
> Ενώ το πλήθος των ελέγχων συνέπειας μας δινει σε κάποιο βαθμό το "κόστος επεξεργασίας" του κάθε κόμβου που επισκεφτήκαμε

Υπολογίζω το πλήθος των constraint checks μετρώντας πόσες φορές καλείται η συνάρτηση constraints() κατά την εκτέλεση, και το πλήθος των visited nodes με χρήση του μετρητή nassigns της κλάσσης csp

iii) Πειραματική μελέτη-σύγκριση των αλγορίθμων

Problem	Method	Visited Nodes	Constraint checks	Runtime (sec)	Result
2-f24	FC + dom/wdeg + unordered vals	463	50926	184	SAT
	MAC + dom/wdeg + unordered vals	304	182825	0.308	SAT
	MAC + dom/wdeg + lcv	304	196113	0.372	SAT
	FC-CBJ + dom/wdeg	258	21559	0.085	SAT
2-f25	FC + dom/wdeg + unordered vals	135709	27804545	56.166	UNSAT
	MAC + dom/wdeg + unordered vals	29392	60136100	101.842	UNSAT
	MAC + dom/wdeg + lcv	29392	60980406	103.936	UNSAT
	FC-CBJ + dom/wdeg	3437	521919	1.507	UNSAT
3-f10	FC + dom/wdeg + unordered vals	122229	15779509	27.605	SAT
	MAC + dom/wdeg + unordered vals	730	1112175	1.874	SAT
	MAC + dom/wdeg + lcv	730	1145078	1.995	SAT
	FC-CBJ + dom/wdeg	8746	1155995	5.244	SAT
3-f11	FC + dom/wdeg + unordered vals	1814304	320775243	485.478	UNSAT
	MAC + dom/wdeg + unordered vals	9352	25840580	49.618	UNSAT
	FC-CBJ + dom/wdeg	258702	42060429	109.890	UNSAT
8-f10	FC + dom/wdeg + unordered vals	-	-	> 5 min	SAT
	MAC + dom/wdeg + unordered vals	9540	19080700	43.249	SAT
	FC-CBJ + dom/wdeg	35447	4453000	27.865	SAT
8-f11	FC + dom/wdeg + unordered vals	204922	42162343	68.855	UNSAT
	MAC + dom/wdeg + unordered vals	44034	111992485	226.730	UNSAT
	FC-CBJ + dom/wdeg	13791	2914120	9.556	UNSAT
14-f27	FC + dom/wdeg + unordered vals	-	-	> 5 min	SAT
	MAC + dom/wdeg + unordered vals	11302	4406353	22.927	SAT
	FC-CBJ + dom/wdeg	20911	1306706	23.468	SAT
14-f28	FC + dom/wdeg + unordered vals	-	-	-	-
	MAC + dom/wdeg + unordered vals	-	-	> 30 min	UNSAT
	FC-CBJ + dom/wdeg	1480	97516	1.765	UNSAT
6-w2	FC + dom/wdeg + unordered vals	253	49661	0.066	UNSAT
	MAC + dom/wdeg + unordered vals	44	93922	0.110	UNSAT
	FC-CBJ + dom/wdeg	253	46348	0.075	UNSAT
7-w1-f4	FC + dom/wdeg + unordered vals	31130	1183368	4.683	SAT
	MAC + dom/wdeg + unordered vals	480	345129	0.421	SAT
	FC-CBJ + dom/wdeg	1900	105911	0.523	SAT
7-w1-f5	FC + dom/wdeg + unordered vals	-	-	> 5 min	UNSAT
	MAC + dom/wdeg + unordered vals	9028	33373978	27.786	UNSAT
	FC-CBJ + dom/wdeg	5243	346197	1.248	UNSAT
11	FC + dom/wdeg + unordered vals	6228	1409893	8.086	SAT
	MAC + dom/wdeg + unordered vals	3221	6828125	13.992	SAT
	FC-CBJ + dom/wdeg	6228	1249953	9.086	SAT

> Η υλοποίηση του dom/wdeg που έχω κάνει, όπως θα δείτε, δεν έχει κάποιο στοιχείο τυχαιότητας, οπότε όσες φορές και να τρέξει θα επιστρέψει το ίδιο αποτέλεσμα. Συνεπώς έχω

γράψει στον πίνακα τα αποτελέσματα μιας εκτέλεσης μόνο. (Στον κώδικα μου, οι δοκιμές γίνονται απο default 5 φορές για κάθε περίπτωση, για όταν θέλω να δοκιμάσω με χρήση mrv ή να τρέξω τον min_conflicts)

> Έκανα δοκιμές και με mrv, ο οποίος βελτίωσε την απόδοση από τον first_unassigned_variable ελάχιστα, και δεν μπορούσε ούτε αυτός να βγάλει αποτέλεσματα σε πρακτικούς χρόνους πέρα από το πρόβλημα 2-f24. Η υλοποίηση του dom/wdeg έφερε πραγματικά εντυπωσιακή βελτίωση στην απόδοση, με πολύ μικρές αλλαγές, και έκανε "βιώσιμη" την λύση σχεδόν όλων των προβλημάτων (εκτώς του 14-f28 που θέλει πανω από 1 ώρα), και έκανε εμφανή και στην πράξη την τεράστια σημασία της σειράς των μεταβλητών σε ένα πρόβλημα ικανοποίησης περιορισμών. Η προσέγγιση "λύσε το δύσκολο κομμάτι πρώτα¨ του dom/wdeg του δίνει ένα τεράστιο πλεονέκτημα σε δύσκολα ή μη ικανοποιήσημα προβλήματα.

> Σε περιπτώσεις που κάποια δοκιμή αργούσε πολύ να επιστρέφει αποτέλεσμα την σταματούσα και έβαζα παύλα στον πίνακα

> Παρατηρούμε ότι η χρήση lcv γενικά οδηγεί σε μικρή αύξηση μόνο του πλήθους constraint checks που τελικά οδηγεί σε μικρές αυξήσεις στον τελικό χρόνο εκτέλεσης. Απ ότι φαίνεται δεν είναι ο καταλληλότερος τρόπος επιλογής values για το πρόβλημα μας.

> Στον παραπάνω πίνακα βλέπουμε ότι φαίνεται να υπερισχύει σαν μέθοδος ο MAC μαζί με τον FC-CBJ, καθώς με "κόστος" περισσότερα constraint checks, επιτυγχάνουν σημαντικά χαμηλότερο πλήθος visited nodes, γεγονός που του δίνει το πάνω χέρι σε ιδιαίτερα σύνθετα προβλήματα, ή προβλήματα χωρίς λύση. Ο FC μπορεί να "κερδίζει¨με μικρή διαφορά για απλά προβλήματα, όμως για πιο σύνθετα, ή προβλήματα χωρίς λύση χρειάζεται τάξεις μεγέθους περισσότερο χρόνο από τον MAC και τον FC-CBJ.

> Η Απόδοση του FC-CBJ είναι πραγματικά εντυπωσιακή, ειδικά σε περιπτώσεις όπως η 14-f28 στην οποία επέστρεψε αποτέλεσμα σχεδόν αμεσα ενώ οι άλλες μέθοδοι θέλαν αδιανόητο πολύ χρονο. (Η απόδοση είναι τόσο καλή που σχεδόν ξεπερνάει τις προσδοκίες που είχα από την θεωρία)

> Σε πολύ μεγάλο βαθμό, τα αποτελέσματά μου συμφωνούν με τα αναμενόμενα αποτελέσματα της θεωρίας (Βλέπε "Αξιολόγηση Αλγορίθμων Υπαναχώρησης" διαφάνειες 82-84, csp1spp.pdf του μαθήματος) όσον αφορά την σύγκριση μεταξύ των μεθόδων, σύμφωνα με τα κριτήρια σύγκρισης που ανέφερα παραπάνω.

> Σε γενικές γραμμές για το συγκεκριμένο πρόβλημα, καλύτερη μέθοδος φαίνεται να είναι ο MAC ή ο FC-CBJ σε συνδυασμό με dom/wdeg καθώς όχι μονο μας δίνουν τα κατα μέσο όρο ταχύτερα αποτελέσματα μεταξύ όλων των περιπτώσεων, αλλά ταυτόχρονα μπορούν εντοπίζουν ότι ένα πρόβλημα είναι UNSAT πολύ γρήγορα. (Νομιζω ο FC-CBJ τελικά κερδιζει :) τουχάχιστον για το πρόβλημα RFLA)

> Όσον αφορά τον FC-CBJ, η λογική του είναι η εξής:
Για κάθε μεταβλητή που διαλέγει, της αναθέσει τιμή, και στη συνέχεια γίνεται έλεγχος αν η συγκεκριμένη ανάθεση προκαλεί αποκοπές σε domains μελλοντικών μεταβλητών που να οδηγούν σε domain wipeout(FC functionality). Σε περίπτωση εντοπισμού domain wipeout, ελέγχει ποιές από τις μέχρι τώρα assigned μεταβλητές επηρέασαν το domain που άδειασε, και τις προσθέτει στο conflict set της current μεταβλητής. Αν δεν εντοπιστεί domain wipeout, τότε απλά συνεχίζει

αναδρομικά να προσπαθήσει να κάνει assign και αλλες μεταβλητές. Αν καμία από τις διαθέσιμες τιμές στο current domain της current μεταβλητής δεν καταφέρει να οδηγήσει στη λύση για το δεδομένο assignment, τότε πρέπει να γίνει backjump στο στην πιο πρόσφατη μεταβλητή στο conflict set της current μεταβλητης ώστε να δοκιμαστεί διαφορετικό assignment. Συνεπώς, αρχικά ελέγχεται αν κάποια από τις assigned μεταβλητές έχει επηρεάσει το domain της current, και προστίθεται ανάλογα στο conflict set της current. Τέλος, παίρνουμε το τελευταίο στοιχείο του conf set, και κάνουμε merge το current conflict set με το conflict set της μεταβλητης στην οποία θα πηδήξουμε, ώστε να διατηρήσουμε την πληροφορία. Κατά το jump, καθαρίζουμε - επαναφέρουμε τις αλλαγές που έχουν γίνει από όσες μεταβλητές χρειαστεί να κάνουμε unassign. Στη υλοποίηση μου βασίστηκα στο kondrak-thesis.pdf και το αρχικό paper του Patrick Prosser[Forward checking with backmarking:1993]

iv) Η τυχαιότητα που εισάγει στην μέθοδο επίλυσης η mrv οδηγεί σε μεταβλητότητα της απόδοσης των μεθόδων μας μεταξύ εκτελέσεων, γεγονός που δείχνει στην πράξη μια ακόμα φορά, πόσο μεγάλη σημασία έχει να είναι "καλό" το Variable ordering για επίτευξη καλών επιδόσεων σε ένα csp.

v) Για πιο συγκεκριμένο σχολιασμό οσον αφορά την λειτουργία του κώδικα, μπορείτε να διαβάσετε τα σχόλια στα παραπάνω αρχεία πηγαίου κώδικα.

4.

Πειραματίστηκα για διαφορα μεγέθη max_steps, από το default(100000) μέχρι και 1000, και σε γενικές γραμμές κατέληξα στο συμπέρασμα ότι πιο μεγάλη σημασία για την αποτελεσματικότητα του min-conflicts έχει ο παράγοντας της τύχης που εισάγεται στον αλγόριθμο κατά τις τυχαίες επιλογές που κάνει. Θεωρητικά και με βάση την λογική, όσο μεγαλύτερο πλήθος βημάτων επιτρέψουμε στον min-conflicts, τόσο πιο πιθανό είναι να βρεί μια λύση, παρόλα αυτά, στην πράξη βλέπουμε ότι η πιθανότητα αυτή δεν αυξάνεται σημαντικά, και στο τέλος, υπερισχύει και μας περιορίζει η χρονική πολυπλοκότητα του αλγορίθμου. Αφού λοιπόν δεν έχουμε ιδιαίτερο κέρδος από μεγαλύτερο πλήθος βημάτων, κατά την κύρια μελέτη του min-conflict που διεξήγαγα, χρησιμοποίησα ως max_steps το 1000. (Όταν εβρισκε λύση την έβρισκε σε περίπου 250-500 steps συνήθως οπότε τα 1000 ήταν ένα καλό μεγεθος, και οι χρόνοι εκτέλεσης παρέμειναν σύντομοι)

Ακόμα, λόγω της αναξιόπιστης-τυχαίας αυτής φύσης του, ο min-conflicts δεν είναι ο καλύτερος τρόπος να προσεγγίσουμε προβλήματα περιορισμών τα οποία δεν γνωρίζουμε από πριν ότι έχουν λύση, καθώς μπορεί να πάρει πολύ χρόνο μέχρι να βρεθεί η λύση αν υπάρχει, ή να πάρουμε "αρκετα" UNSAT ώστε να μπορούμε να υποθέσουμε ότι δεν υπάρχει. (Καμια εγγύηση όμως ότι δεν υπάρχει, μπορεί απλά να είμασταν άτυχοι)

Επιπλέον, είναι λογικό, όσο μεγαλύτερο και πιο σύνθετο είναι ένα csp, τόσο πιο αναξιόπιστος να είναι ο min-conflicts, καθώς για την επίλυση του θα πρέπει να είμαστε "εξαιρετικά τυχεροί". (Μεγαλύτερο πλήθος πιθανών επιλογών => μικρότερη πιθανότητα να βρούμε την "σπάνια" περίπτωση της λύσης)

Γενικά, παρατηρώντας τα αποτελέσματα από τις εκτελέσεις του min-conflicts για το πρόβλημα μας, τόσο για το ίδιο μέγεθος max_steps, όσο και για διαφορετικά μεγέθη max_steps, γίνεται εμφανές ότι ο min-conflicts δεν είναι ιδιαιτερα κατάλληλος για το πρόβλημα μας.

Μεταξύ αρκετών εκτελέσεων, ο min-conflicts, επιστρέφει σπάνια λύση (SAT μόνο για το 1~2% περίπου των εκτελέσεων για max_steps=1000), καθώς στην πράξη το αν θα βρεί λύση εξαρτάται από το πλήθος των steps που έχει διαθέσιμα, το οποίο όμως περιορίζεται σημαντικά από την εν μέρει "τυχαία" φύση του min-conflicts η οποία υπερισχύει. Για να επιτύχουμε μεγαλύτερη αξιοπιστία του αλγορίθμου μπορούμε να αυξήσουμε το μέγεθος max_steps, το οποίο όμως θα οδηγήσει σε τεράστιες χρονικές καθυστερήσεις, ή εναλλακτικά να τρέξουμε μεγάλο πλήθος εκτελέσεων με μικρό μέγεθος max_steps(~300-400). Η δεύτερη αυτή μέθοδος οδηγεί στα καλύτερα αποτελέσματα που πέτυχα για τον αλγόριθμο αυτό, τουλάχιστον απο πλευράς χρονικής ταχύτητας εύρεσης λύσης(Με 100 εκτελέσεις για max_steps=300). Παρόλα αυτά, είναι εμφανές ότι χωρίς περαιτέρω βελτιώσεις, ο min-conflicts δεν είναι η καλύτερη επιλογή για το πρόβλημα μας. Συνεπώς η χρήση των μεθόδων που εξετάσαμε στο ερώτημα 3, είναι σημαντικά καλύτερη οσον αφορά την λύση του RLFA.

Όπως ζήτησε στο μάθημα ο κύριος κουμπαράκης, ως μπόνους, πρόσθεσα στην υλοποίηση μου και την δυνατότητα εκτύπωσης του πλήθος των συγκρούσεων που απομένουν όταν σταματάει ο min-conflicts λόγω ολοκλήρωσης των max_steps, ώστε να μπορούμε να δούμε πόσο "κοντα" έφτασε ο min-conflicts στη λύση του csp.

Με βάση τις τιμές που παίρνουμε από την δυνατότητα αυτή, παρατηρούμε ότι αρκετές φορές ο min conflicts φτάνει πολύ κοντά σε λύση (2-4 conflicts) αλλά παρόλα αυτά αποτυγχάνει. Τις περισσότερες φορές όμως δεν καταφέρνει να πλησιάσει τόσο. Για καλύτερη εικόνα της συμπεριφοράς του, τρέξτε τον για TEST_AMOUNT=100 και MAX_STEPS=300.

```
C:\Users\john\Desktop\3_aihw2020>python main.py
```

```
                        - AI Assignment 3 -
                    Constraint satisfaction problems
                    Radio link frequency assignment
```

```
  (i) Reading Input Files...
  > Reading: ./rlfap/var2-f24.txt
  > Reading: ./rlfap/dom2-f24.txt
  > Reading: ./rlfap/ctr2-f24.txt
  < Input Complete >
```

```
  (i) Testing using MIN-CON
```

```
  > Testing on problem:        2-f24
  > Amount of test-runs:       5
  > Min-conflicts max-steps:   300
```

Test	Visited nodes	Constraint checks	Runtime	Result
1	500	917630	0.841	UNSAT
2	264	238090	0.208	SAT
3	500	919158	0.892	UNSAT
4	500	923748	0.910	UNSAT
5	500	928470	0.838	UNSAT
Average	452.80	785419.20	0.738	---

```
  (!) Remaining conflicts when min-conflicts was terminated:
```

1	18 conflicts remaining
2	0 conflicts remaining
3	22 conflicts remaining
4	16 conflicts remaining
5	16 conflicts remaining

```
  (!) All tests completed without errors
```

Πρόβλημα 2:

Η επίπλωση του δωματίου με τα δεδομένα έπιπλα υπό τις δεδομένες συνθήκες, είναι δυνατή, και μάλιστα με παραπάνω από έναν τρόπους. Έναν ενδεικτικό τρόπο / παράδειγμα μπορείτε να δείτε στην διπλανή εικόνα. (Έχει σχεδιαστεί με αναλογία 1 εκατοστό = 1 pixel)

Όμως, η λύση αυτή βρέθηκε από έναν άνθρωπο, με βάση την αντίληψη που είχει για τα αντικείμενα, τον ίδιο τον χώρο, τις διαστάσεις του και τις προσωπικές του προτιμήσεις και εμπειρίες. Όπως είναι εμφανές, κάτι τέτοιο είναι πολύ δύσκολο να γίνει από ένα πρόγραμμα. Για να μπορούμε λοιπόν να λύνουμε αξιόπιστα (αν υπάρχουν λύσεις) ένα πρόβλημα τέτοιου τύπου προγραμματιστικά, ένας καλός τρόπος είναι να το αναπαριστούμε ως ένα πρόβλημα ικανοποίησης περιορισμών.

Στην συγκεκριμένη περίπτωση το πρόβλημα μας μπορεί να αναπαρασταθεί ως πρόβλημα ικανοποίησης περιορισμών ως εξής:

Μεταβλητές:
Καναπές, Γραφείο, Καρέκλα, Κρεβάτι

Πεδίο: (Αν έχουμε αναπαραστήσει το δωμάτιο με σύστημα καρτεσιανών συντεταγμένων)
{ Ζευγάρι ακεραίων x,y = συντεταγμένες της πάνω δεξιάς γωνίας (χωρις βλαβη γενικότητας) του κάθε επίπλου με βάση την οποία και τη σταθερή δεδομενη τριπλέτα ακεραίων h,w,l (= διαστάσεις των πλευρών του), προκύπτει η θεση της επιφάνειας επίπλου στο δωμάτιο } - Πεδίο κοινό για όλες τις μεταβλητές
(Θα μπορούσαμε να έχουμε και rotation αλλά απλά περιπλέκει την αναπαράσταση περισσότερο οπότε μας είπανε στο piazza να μην την συμπεριλάβουμε.)

Περιορισμοί: (Όλοι ο έλεγχοι θέσης γίνονται με αναπαράσταση των επίπλων από την άνω δεξια τους γωνία σε συνδυασμό με τα μεγέθη των πλευρών)

// Τοποθεσία εσωτερικά της διαθέσιμης επιφάνειας του δωματίου => Όλα τα έπιπλα
C(Καναπές, Γραφείο, Καρέκλα, Κρεβάτι) = { Ελέγχοντας πότε ένα έπιπλο είναι τοποθετημένο στη διαθέσιμη επιφάνεια (όχι πάνω σε τοιχο, ούτε εκεί που ανοίγει η πόρτα) παράγουμε το σύνολο όλων των διαθέσιμων συνδυασμών "νόμιμων" τοποθεσιών = Το σύνολο αυτού του περιορισμού }

// Όχι επαφή/επικάλυψη => Όλα τα έπιπλα
C(Καναπές, Γραφείο, Καρέκλα, Κρεβάτι) = { Ελέγχοντας σε ποιές περιπτώσεις έχουμε επαφή επίπλων, παράγουμε το σύνολο όλων των δυνατών συνδυασμών τοποθεσιών των επίπλων στο δωμάτιο ώστε να **μην υπάρχει επαφή**= Το σύνολο αυτού του περιορισμού }

(Αφού υπάρχει ο περιορισμός ότι τα έπιπλα δεν μπορουν να επικαλύπτονται (να τοποθετούνται το ένα πάνω στο άλλο) αγνόησα πλήρως το ύψος των επίπλων στο πρόβλημα μας)

// Γραφείο κοντα στο παράθυρο => Μόνο το γραφείο
C(Γραφείο) = { Ελέ γχοντας όλες τις δυνατές τοποθεσίες του γραφείου κρατάμε στο σύνολο αυτό, μόνο αυτές στις οποίες κάποια από τις πλευρές του επίπλου βρίσκεται κοντινότερα από 100 εκατοστά από την πηγή φωτος - το παράθυρο (100 εκ χωρίς βλάβη της γενικότητας) = Το σύνολο αυτού του περιορισμού }

Έχοντας λοιπόν πλέον ορίσει πλήρως το πρόβλημα, και γνωρίζοντας τα σύνολα των περιορισμών, μπορούμε να συνδυάσουμε με διάφορους τρόπους - αναζητήσουμε τα περιεχόμενα των συνόλων ώστε να ικανοποιούνται ταυτόχρονα όλοι οι περιορισμοι, και κατα συνέπεια να οδηγηθούμε σε μια ή περισσότερες λύσεις.

Πρόβλημα 3:
1. Μπορούμε να μοντελοποιήσουμε το πρόβλημα μας ως πρόβλημα ικανοποίησης περιορισμών, ορίζοντάς το ως εξής.

Μεταβλητές: A1, A2, A3, A4, A5

Πεδίο: // Η ώρα εκκίνησης μιας ενέργειας
{ 9:00, 10:00, 11:00 } - Πεδίο κοινό για όλες τις μεταβλητές
(Για την A4 γίνεται τελικά { 9:00, 11:00 }, βλέπε τελευταίο περιορισμό)

Περιορισμοί: // Έστω T(x) η ώρα εκκίνησης μιας ενέργειας

// Η A1 να ξεκινάει μετά την A3
C(A1, A3) = { Το σύνολο των δυνατών αναθέσεων των εργασιών A1...A5 έτσι ώστε να ικανοποιείται η σχέση: T(A1) > T(A3) }

// Η A3 ξεκινάει πριν την A4
C(A3, A4) = { Το σύνολο των δυνατών αναθέσεων των εργασιών A1...A5 έτσι ώστε να ικανοποιείται η σχέση: T(A3) < T(A4) }

// Η A3 ξεκινάει μετά την A5
C(A3, A4, A5) = { Το σύνολο των δυνατών αναθέσεων των εργασιών A1...A5 έτσι ώστε να ικανοποιείται η σχέση: T(A5) < T(A3)

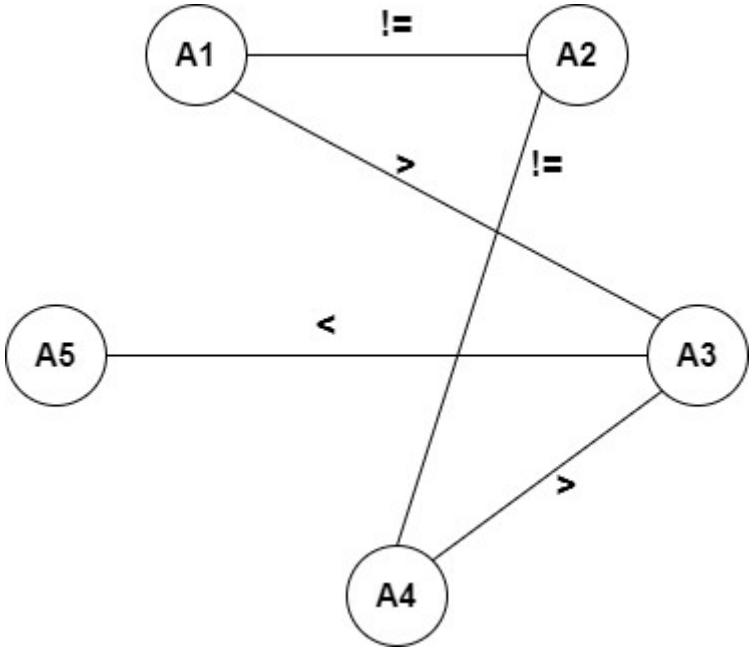
// Η A2 δεν μπορεί να εκτελείται την ίδια ώρα με την A1
C(A2, A1) = { Το σύνολο των δυνατών αναθέσεων των εργασιών A1...A5 έτσι ώστε να ικανοποιείται η σχέση: T(A2) != T(A1) }

// Η A2 δεν μπορεί να εκτελείται την ίδια ώρα με την A4
C(A2, A1) = { Το σύνολο των δυνατών αναθέσεων των εργασιών A1...A5 έτσι ώστε να ικανοποιείται η σχέση: T(A2) != T(A4) }

// Η A4 δεν μπορεί να αρχίσει στις 10:00
C(A4) = { Το σύνολο των δυνατών αναθέσεων των εργασιών A1...A5 έτσι ώστε να ικανοποιείται η σχέση: T(A4) != 10:00 } (Για να απλοποιήσω το πρόβλημα στο σημείο αυτό, μπορούμε να αφαιρέσουμε πλήρως αυτόν τον περιορισμό, και στη θέση του, απλά να μετατρέψουμε το πεδίο

ορισμού της ενέργειας Α4 από { 9:00, 10:00, 11:00 }, σε { 9:00, 11:00 }. Με βάση την παραδοχή αυτή σχεδιαζω και παρακάτω τον γράφο περιορισμών)

2. Με βάση τη θεωρία του μαθήματος, ο γράφος περιορισμών του προβλήματος μας είναι ο εξής:



3. Εφαρμόζοντας τον αλγόριθμο AC-3 στο πρόβλημα μας μέχρι αυτό να γίνει συνεπές ώς προς τις ακμές έχουμε την εξής διαδικασία:

Αναλυτικά τα domains (Πεδία τιμών), από το ερώτημα 1 της άσκησης:
D(A1, A2, A3, A5) = { 9:00, 10:00, 11:00 }
D(A4) = { 9:00, 11:00 }

Με βάση την θεωρία για τον AC-3 καθώς και τα παραδείγματα που κάναμε στο φροντιστήριο, έχουμε:
(Ξεκινάμε τις αναθέσεις τιμών στις μεταβλητές μας)

> **A1 = 9:00**, Λόγω των περιορισμών, επηρεάζονται τα domains των κόμβων "γειτόνων" στον γράφο περιορισμών => D(A2)={ 10:00, 11:00 }, **D(A3) = {} => Μη συνεπής => Επόμενη τιμή για A1**

> **A1 = 10:00** => D(A2)={ 9:00, 11:00 }, D(A3)={ 9:00 } άρα πρέπει να ελέγξουμε τις ακμές των γειτόνων που πιθανόν επηρεάζονται. (A4, A2), (A5, A3), (A4, A3):
(A4, A2) = Μη συνεπής => D(A4)={ 10:00, 11:00 } => εξετάζουμε και την (A3, A4)
(A5, A3) = Μη συνεπής => **D(A5)={} => Επόμενη τιμή για A1**

> **A1 = 11:00** => D(A2)={ 9:00, 10:00 }, D(A3)={ 9:00, 10:00 } ελέγχουμε τις ακμές των γειτόνων
(A4, A2)= OK
(A5, A3)= Μη συνεπής => **D(A5)={9:00}** ελέγχουμε την (A3, A5)
(A4, A3)= Μη συνεπής => **D(A4)={11:00}**, ελέγχουμε την (A2, A4)
(A3, A5)= Μη συνεπης => **D(A3)={10:00}** οκ
(A2, A4)= OK => **D(A2)={ 9:00, 10:00 }**

Άρα τελικά καταλήξαμε σε μία ανάθεση τιμών η οποία κάνει το πρόβλημα μας συνεπές, δηλαδή σε μία λύση (στην περίπτωση μας καταλήξαμε σε δύο δυνατές λύσεις) του συγκεκριμένου προβλήματος περιορισμών (= δυνατός συνδυασμός τιμών τέτοιος ώστε να ικανοποιούνται όλοι οι προσδιορισμοί)

Συγκεκριμένα οι λύσεις που βρήκαμε είναι οι εξείς:
Η Α1 εκτελείται στις 11:00
Η Α2 εκτελέιται ή στις 9:00, ή στις 10:00
Η Α3 εκτελείται στις 10:00
Η Α4 εκτελείται στις 11:00
Η Α5 εκτελείται στις 9:00

Σημαντικά σχόλια:
1. Το development έγινε εξ ολοκλήρου σε υπολογιστή με windows 10, καθώς οι υπολογιστές στη σχολή δεν έχουν εγκατεστημένα τα sortedcontainers.
2. Για οποιοδήποτε πρόβλημα, απορια, διευκρίνηση ή πρότασεις για βελτίωση των υλοποιήσεων και των μεθόδων μου, μπορείτε να έρθετε σε επικοινωνία μαζί μου στο sdi1800164@di.uoa.gr