Πρόβλημα 1:

Στους πίνακες παρακάτω τα instances εκτελέστηκαν με τις ευρετικές **dom/wdeg** για επιλογή μεταβλητής και **lcv** για επιλογή τιμής. **Time**: Χρόνος εκτέλεσης σε δευτερόλεπτα, **Assignments**: πόσες φορές μια μεταβλητή πήρε τιμή, **Constraint Checks**: πόσες φορές ελέγχθηκε αν παραβιάζεται ο περιορισμός ανάμεσα σε 2 γειτονικές μεταβλητές. Φαίνοντα τα στατιστικά για όσα instances βρέθηκε λύση. **SAT** σημαίνει ότι υπάρχει ανάθεση τιμών στις μεταβλητές, τέτοια ώστε να μην παραβιάζονται οι περιορισμοί. **UNSAT** σημαίνει ότι **δεν** υπάρχει ανάθεση τιμών στις μεταβλητές, τέτοια ώστε να μην παραβιάζονται οι περιορισμοί. Σε κάποια instances δε βρήκα λύση με όλους τους αλγορίθμους. Τον αλγόριθμο min conflicts, τον εκτέλεσα όπως ήταν και δε βρήκα λύση σε κανένα instance, ούτε κατέληξα σε κάποια τροποποίηση που να βρίσκει λύση. Αυτό δε σημαίνει ότι ο αλγόριθμος είναι κακός, απλά δεν είναι κατάλληλος για το πρόβλημα μας, διότι απλά βάζει τιμές σε όλες τις μεταβλητές βάσει των αριθμό των conflicts και μετά αν αυτό δεν αποφέρει λύση, τότε διαλέγει εντελώς τυχαία μεταβλητές, χωρίς κάποια ευρετική ή κάποιο κριτήριο, και τους δίνει καινούριες τιμές, ελέγχει αν βρήκε λύση και ξανακάνει την ίδια διαδικασία. Στο πρόβλημα μας, έχουμε πολυάριθμες μεταβλητές, με μεγάλα domains, οπότε χρειάζεται να επιλέγουμε έξυπνα μεταβλητές και τιμές, βάσει κάποιων κριτηρίων.

Instance 11 (SAT)	**Time (sec)**	**Assignments**	**Constraint Checks**
mac	15.46	3238	6890172
	15.661	3238	6890172
	13.836	3238	6890172
	13.725	3238	6890172
	18.185	3238	6890172
Average	**15.3734**	**3238**	**6890172**
fc	7.085	6315	1391953
	6.548	6315	1391953
	6.297	6315	1391953
	6.684	6315	1391953
	9.106	6315	1391953
Average	**7.144**	**6315**	**1391953**
fc-cbj	9.705	6315	4440306
	8.297	6315	4440306
	8.407	6315	4440306
	7.91	6315	4440306
	7.872	6315	4440306
Average	**8.4382**	**6315**	**4440306**

Instance 2-f24 (SAT)	Time (sec)	Assignments	Constraint Checks
mac	0.347	324	188614
	0.42	324	188614
	0.383	324	188614
	0.352	324	188614
	0.329	324	188614
Average	**0.3662**	**324**	**188614**
fc	0.16	487	39119
	0.157	487	39119
	0.184	487	39119
	0.153	487	39119
	0.144	487	39119
Average	**0.1596**	**487**	**39119**
fc-cbj	0.08	260	33865
	0.079	260	33865
	0.075	260	33865
	0.076	260	33865
	0.073	260	33865
Average	**0.0766**	**260**	**33865**

Instance 3-f10 (SAT)	Time (sec)	Assignments	Constraint Checks
mac	2.118	819	1165560
	2.374	819	1165560
	2.234	819	1165560
	2.076	819	1165560
	2.31	819	1165560
Average	**2.2224**	**819**	**1165560**
fc	21.637	91288	11463233
	23.056	91288	11463233
	21.254	91288	11463233
	21.368	91288	11463233
	22.415	91288	11463233
Average	**21.946**	**91288**	**11463233**
fc-cbj	11.783	19783	13213758
	11.684	19783	13213758
	11.21	19783	13213758
	11.628	19783	13213758
	11.384	19783	13213758
Average	**11.5378**	**19783**	**13213758**

Instance 7-w1-f4 (SAT)	Time (sec)	Assignments	Constraint Checks
mac	0.492	480	346398
	0.516	480	346398
	0.45	480	346398
	0.425	480	346398
	0.442	480	346398
Average	**0.465**	**480**	**346398**
fc	7.089	46418	1867727
	6.664	46418	1867727
	6.022	46418	1867727
	6.017	46418	1867727
	5.867	46418	1867727
Average	**6.3318**	**46418**	**1867727**
fc-cbj	1.268	1755	2349521
	1.292	1755	2349521
	1.267	1755	2349521
	1.348	1755	2349521
	1.336	1755	2349521
Average	**1.3022**	**1755**	**2349521**

Instance 8-f10 (SAT)	Time (sec)	Assignments	Constraint Checks
mac	130.371	31464	50262048
	124.455	31464	50262048
	116.826	31464	50262048
	113.763	31464	50262048
	113.281	31464	50262048
Average	**119.7392**	**31464**	**50262048**
fc-cbj	77.622	54252	160486926
	86.716	54252	160486926
	90.255	54252	160486926
	88.795	54252	160486926
	85.062	54252	160486926
Average	**85.69**	**54252**	**160486926**

Instance 14-f27 (SAT)	Time (sec)	Assignments	Constraint Checks
fc-cbj	28.827	21933	28650440
	32.624	21933	28650440
	33.622	21933	28650440
	35.435	21933	28650440
	32.183	21933	28650440
Average	**32.5382**	**21933**	**28650440**

Instance 2-f25 (UNSAT)	Time (sec)	Assignments	Constraint Checks
mac	117.905	26804	58400689
fc	48.095	130277	22785778
fc-cbj	54.506	127983	35114909

Instance 3-f11 (UNSAT)	Time (sec)	Assignments	Constraint Checks
mac	60.257	9619	28701487
fc	162.337	615910	88364473
fc-cbj	173.147	322024	164385610

Instance 8-f11 (UNSAT)	Time (sec)	Assignments	Constraint Checks
mac	90.564	18207	38489542
fc	1182.17	2981856	463704015
fc-cbj	29.142	40782	21348627

Instance 14-f28 (UNSAT)	Time (sec)	Assignments	Constraint Checks
mac	56.021	18000	12655272
fc	74.747	105200	6711790
fc-cbj	15.197	14983	8615741

Instance 6-w2 (UNSAT)	Time (sec)	Assignments	Constraint Checks
mac	0.141	44	93188
fc	0.084	295	51409
fc-cbj	0.094	295	77878

Κοιτώντας τους παραπάνω πίνακες, παρατηρώ ότι ο αλγόριθμος FC-CBJ μάλλον είναι ο καλύτερος για τα instances του συγκεκριμένου προβλήματος. Αυτό προκύπτει, αν σκεφτούμε ότι βρίσκει τη λύση σε όλα τα instances για τα οποία υπάρχουν συνδυασμοί τιμών και μεταβλητών που δεν παραβιάζουν τους περιορισμούς, ενώ οι υπόλοιποι όχι. Επίσης, στα περισσότερα instances, ο χρόνος εκτέλεσης με FC-CBJ, είναι καλύτερους σε σχέση με τους άλλους αλγόριθμους. Θεωρώ πως το καλύτερο και ταυτόχρονα σημαντικότερο κριτήριο σύγκρισης για τους αλγόριθμους, είναι η αποτελεσματικότητα και ο χρόνος εκτέλεσης. Όσον αφορά τα assignments και τα constraint checks, μπορεί σε κάποια instances να είναι

μεγαλύτερος ο αριθμός τους για τον FC-CBJ, αλλά αυτό δεν έχει καμία σημασία εφόσον βρίσκει λύση (είτε αυτή η λύση είναι SAT είτε UNSAT) ο αλγόριθμος ταχύτερα.

Σχόλια για τον κώδικα:

Το αρχείο προς εκτέλεση είναι το main.py

Τα απαραίτητα αρχεία είναι τα **search.py, utils.py, findPairs.py και main.py**

Στο αρχείο **findPairs.py**, αρχικά αντέγραψα, από το αρχείο που μας δόθηκε στο github, την κλάση CSP και τις απαραίτητες συναρτήσεις που χρειάζεται για να λειτουργήσει και για να τρέξουν οι αλγόριθμοι. Για την ανάγνωση των αρχείων και τη δημιουργία των απαραίτητων δομών, υλοποίησα τις εξής συναρτήσεις: **OpenVar**, η οποία ανοίγει τα αρχεία με prefix “var”, διαβάζει μία μία τις γραμμές του αρχείου και δημιουργεί κι επιστρέφει μια λίστα από tuples, όπου κάθε tuple έχει για 1ο στοιχείο μια μεταβλητή και για 2ο στοιχείο τον αριθμό του domain που της αντιστοιχεί. **OpenDom**, η οποία ανοίγει τα αρχεία με prefix “dom”, διαβάζει μία μία τις γραμμές του αρχείου και χρησιμοποιώντας τη λίστα που δημιουργήθηκε προηγουμένως, δημιουργεί κι επιστρέφει ένα dictionary όπου για keys έχει τις μεταβλητές και για values μια λίστα με τις τιμές του domain τους. Για κάθε μεταβλητή του κάθε tuple της λίστας, βρίσκει τον αριθμό του domain που της αντιστοιχεί και διαβάζει τις αντίστοιχες τιμές του domain και τις βάζει σε μια λίστα. Μετά παίρνει αυτήν τη λίστα και τη βάζει για value στη θέση του dictionary με key ίσο με τη μεταβλητή. **OpenCtr**, η οποία ανοίγει τα αρχεία με prefix “ctr”, διαβάζει μία μία τις γραμμές του αρχείου και δημιουργεί κι επιστρέφει δύο dictionaries. Σε κάθε γραμμή, διαβάζει το ζεύγος των μεταβλητών και το βάζει για key υπό τη μορφή tuple στο dictionary constraints και για value βάζει τον περιορισμό (“>” ή “=” και την τιμή του). Για το ίδιο ζεύγος μεταβλητών βάζει στο dictionary neighbors για key τη μία μεταβλητή και για value σε μια λίστα την άλλη μεταβλητή και αντίστροφα. Άρα αν διαβάσει “ A B > 50 “ θα βάλει στο dictionary constraints: constraints[(A,B)] = [“>”, “50”] και στο neighbors: neighbors[A] = [B] και neighbors[B] = [A]. Έτσι, το dictionary constraints έχει για keys tuples με τα ζεύγη των μεταβλητών και values μια λίστα με τα στοιχεία του περιορισμού υπό μορφή string και το dictionary neighbors έχει για keys τις μεταβλητές και για values μια λίστα με τους γείτονες τους.

Για την υλοποίηση της **dom/wdeg**, πρόσθεσα μία έξτρα δομή στην κλάση CSP, την **weights**, ένα dictionary που για κάθε ζεύγος γειτόνων A, B έχει για keys tuples (A,B) και (B,A) και για values μια τιμή, η οποία εκφράζει το βάρος που έχει η B στην A και η A στην B αντίστοιχα. Τα βάρη είναι αρχικοποιημένα σε 1 και κάθε φορά που συμβαίνει domain wipeout π.χ. σε μια μεταβλητή A λόγω μιας μεταβλητής B τότε αυξάνω το βάρος που έχει η B στην A (αυτό πραγματοποιείται με τοποθέτηση ελέγχου στις συναρτήσεις που καλούνται από τους αλγορίθμους, πιο συγκεκριμένα στη forward_checking και στη revise που καλείται από την AC3, η οποία χρησιμοποιείται για την εκτέλεση του mac) . Στη συνάρτηση **dom_wdeg**, η οποία παίρνει ως όρισμα το assignment και ένα αντικείμενο csp, προσπελαύνω τη λίστα με τις μεταβλητές που υπάρχει στο csp, και για κάθε μεταβλητή από αυτές που δεν τους έχει ανατεθεί τιμή, υπολογίζω την τιμή **dom**, η οποία εκφράζει το πόσες τιμές

υπάρχουν αυτήν τη στιγμή στο domain της. Ύστερα υπολογίζω την τιμή **wdeg**, η οποία εκφράζει το άθροισμα των βαρών που έχουν οι γείτονες της μεταβλητής, εκείνοι οι οποίοι δεν τους έχει ανατεθεί τιμή, στην μεταβλητή αυτή. Τέλος, υπολογίζω το ratio **dom/wdeg** για τη μεταβλητή αυτή. Αφού υπολογιστούν όλα τα ratio για τις μεταβλητές που δεν τους έχει ανετεθεί τιμή, επιστρέφεται η μεταβλητή εκείνη με το μικρότερο ratio.

Για την υλοποίηση του **FC-CBJ**, ακολούθησα τη λογική της backtracking_search και δημιούργησα μια νέα συνάρτηση, την **backjumping_search**, η οποία παίρνει τα ίδια ορίσματα με την backtracking_search και κάνει την εξής διαδικασία. Καλεί τη συνάρτηση **backjump**, που παίρνει ως όρισμα το **assignment** ένα, αρχικά άδειο, dictionary που έχει μέσα για keys τις μεταβλητές που έχουν πάρει τιμή και για values τις αντίστοιχες τιμές που έχουν πάρει, το **CS** ένα, αρχικά άδειο, dictionary που έχει μέσα για keys τις μεταβλητές και για values τη λίστα με τις μεταβλητές που ανήκουν στο Conflict Set της κάθε μεταβλητής, τη μεταβλητή boolean **backjumpNeeded**, αρχικά ως False, που χρησιμεύει για να γνωρίζουμε αν τη συγκεκριμένη στιγμή χρειάζεται να κάνουμε back jump σε μια προηγούμενη μεταβλητή και τέλος τη μεταβλητή int **backjumpVar**, που εκφράζει τη μεταβλητή στην οποία πρέπει να γίνει το back jump, δηλαδή σε αυτήν που πρέπει να γυρίσουμε. Η συνάρτηση backjump αρχικά ελέγχει αν όλες οι μεταβλητές έχουν πάρει τιμή, πράγμα το οποίο θα σημαίνει ότι έχουμε φτάσει σε κατάσταση στόχου. Αν δεν ισχύει το παραπάνω, διαλέγει μια μεταβλητή **var**, με βάση την ευρετική για τις μεταβλητές που της περνάω ως όρισμα, δημιουργεί το conflict set της μεταβλητής αυτής (μια άδεια λίστα) και μετά βάζει τιμές (**value**) στη μεταβλητή, με βάση την ευρετική για τις τιμές που κι αυτή περνάω ως όρισμα. Για κάθε τιμή καλείται η συνάρτηση inference, δηλαδή ο FC στην περίπτωση μας, όμως με μια μικρή παραλλαγή από τον ήδη υλοποιημένο forward_checking. Εδώ χρησιμοποιείται η συνάρτηση **FCforCBJ**, η οποία έχει την ίδια λειτουργία με την forward_checking, δηλαδή τον κανονικό FC, με μόνη διαφορά ότι αν π.χ. μια μεταβλητή B πάθει domain wipeout, τότε η συνάρτηση επιστρέφει False αλλά και τη μεταβλητή B. Αν η συνάρτηση επιστρέψει **True**, τότε αυτό σημαίνει ότι η τιμή value της var είναι εντάξει προς το παρόν, καλείται αναδρομικά η backjump και ελέγχεται η επόμενη μεταβλητή για να της ανατεθεί τιμή. Αν επιστρέψει **False**, τότε ελέγχω για τη μεταβλητή που έπαθε domain wipeout και επέστρεψε ο FC (**confVar**) με ποιες μεταβλητές, από αυτές που έχουν πάρει τιμή, έχει conflict, δηλαδή παραβιάζεται ο μεταξύ τους περιορισμός και βάζω αυτές τις μεταβλητές στο conflict set της var (var = η μεταβλητή που ελέγχεται τώρα). Αντίστοιχα, ελέγχω με ποιες μεταβλητές, από αυτές που έχουν πάρει τιμή, έχει conflict η var και τις βάζω κι αυτές στο conflict set της var. Αν αποτύχουν όλες οι τιμές της var, δηλαδή επιστρέψει ο FC False για όλες τις τιμές του domain της, τότε πηγαίνω στον conflict set της var και βρίσκω τη μεταβλητή, η οποία βρίσκεται βαθύτερα στο δέντρο αναζήτησης, δηλαδή αυτή που πήρε τιμή πιο πρόσφατα, και τη βγάζω από το conflict set της var. Τώρα πρέπει να γίνει back jump στη μεταβλητή αυτή. Το backjumpNeeded γίνεται True και η backjumpVar παίρνει για τιμή τη μεταβλητή αυτή. Μετά, βάζω στο conflict set της backjumpVar, όλες τις μεταβλητές που υπάρχουν στο conflict set της var (προφανώς εκτός από την ίδια την backjumpVar κι από αυτές που υπάρχουν ήδη). Αδειάζω το conflict set της var και επιστρέφω None, backjumpNeeded, backjumpVar, έτσι ώστε η μεταβλητή που θα γυρίσουμε να έχει τις απαραίτητες πληροφορίες. Αν το conflict

set της var είναι άδειο θα πρέπει να επιστρέψει ο αλγόριθμος στην αμέσως προηγούμενη μεταβλητή. Το back jump επιτυγχάνεται ως εξής: ο αλγόριθμος επιστρέφει None, backjumpNeeded και backjumpVar και γυρνάμε προς τα πίσω, περνώντας μία μία τις μεταβλητές, στις οποίες είχαμε αναθέσει τιμή και περιμέναν ένα result από τις επόμενες τους. Το result αυτό είναι None και έτσι κάθε μεταβλητή από αυτές, ελέγχει αν χρειάζεται back jump και αν αυτό αληθεύει, τότε ελέγχει σε ποια μεταβλητή πρέπει να γίνει το back jump. Αν η μεταβλητή που κάνει τον έλεγχο, είναι αυτή που πρέπει να γίνει το back jump (var = backjumpVar), τότε συνεχίζει ο αλγόριθμος από εκεί που είχε μείνει για αυτήν τη μεταβλητή, δηλαδή συνεχίζει η λούπα διαλογής και ελέγχου τιμών από το domain της, για να της ανατεθεί μια νέα τιμή. Αλλιώς, αν δεν είναι αυτή η μεταβλητή ίση με backjumpVar, τότε βγάζω από τη μεταβλητή την τιμή που της έχει ανατεθεί, αδειάζω το conflict set της και επιστρέφει κι αυτή με τη σειρά της None και ο αλγόριθμος γυρνάει στην προηγούμενη κ.ο.κ. μέχρι να φτάσουμε στην backjumpVar. Για την τοποθέτηση των μεταβλητών στα conflict sets χρησιμοποιώ μια λίστα **varSelectOrder**, η οποία περιέχει τις μεταβλητές που έχουν πάρει τιμή με τη σειρά, κατά την οποία έχουν ελεγχθεί και τη συνάρτηση **addToCS**, η οποία παίρνει ως όρισμα το dictionary με τα conflict sets, μια μεταβλητή var, στης οποίας το conflict set πρέπει να βάλουμε μια μεταβλητή, και μια μεταβλητή a, η οποία πρέπει να μπει στο conflict set. Η addToCs παίρνει τη μεταβλητή a και με βάση τη λίστα varSelectOrder (στην οποία έχει πρόσβαση), τη βάζει στο conflict set της var στην κατάλληλη θέση κι έτσι με αυτόν τον τρόπο, πετυχαίνω τα εξής: οι μεταβλητές είναι τοποθετημένες στο conflict set με τη σειρά κατά την οποία ελέγχθηκαν από τον αλγόριθμο και η τελευταία του conflict set είναι η πιο πρόσφατη στην αναζήτηση, οπότε όταν χρειαστεί να κάνουμε back jump απλά παίρνουμε την τελευταία του conflict set. Σε περίπτωση επιτυχίας ο αλγόριθμος επιστρέφει το assignment με τις μεταβλητές και τις τιμές που τους ανατέθηκαν και σε περίπτωση αποτυχίας ο αλγόριθμος επιστρέφει None.

Τέλος, έχω υλοποιήσει τη συνάρτηση **runFiles**, η οποία παίρνει ως ορίσματα τα ονόματα των αρχείων var, dom και ctr και καλεί τις απαραίτητες συναρτήσεις για να τα ανοίξει και δημιουργεί τις απαραίτητες δομές για το CSP καθώς και τη συνάρτηση constraints, η οποία ελέγχει αν οι 2 μεταβλητές, που περνιούνται ως ορίσματα, έχουν περιορισμό μέσω του dictionary constraints που έχει δημιουργηθεί παραπάνω. Αν έχουν, ελέγχει αν παραβιάζεται ο περιορισμός με τις τιμές που τους έχουν ανατεθεί αντίστοιχα οι οποίες περνιούνται κι αυτές ως ορίσματα. Αν παραβιάζεται ο περιορισμός επιστρέφει False, αλλιώς True. Αν δεν υπάρχει περιορισμός τότε επιστρέφει True. Δημιουργεί ένα αντικείμενο CSP για κάθε αλγόριθμο, του περνάει τις κατάλληλες δομές και μετά εκτελεί τους αλγόριθμους. Για τον **MAC** καλείται η backtracking_search, με inference = mac, για τον **FC** πάλι η backtracking_search με inference = forward_checking και για τον **FC-CBJ** καλείται η backjumping_search με inference = FCforCBJ. Και για τους τρεις αλγόριθμους χρησιμοποιείται η ευρετική dom_wdeg για επιλογή μεταβλητών και η ευρετική lcv για επιλογή τιμών. Πριν και μετά το κάλεσμα των αλγορίθμων καλώ τη συνάρτηση time.time() για να υπολογίσω το χρόνο εκτέλεσης του κάθε αλγόριθμου, υπολογίζοντας τη διαφορά των δύο στιγμιοτύπων. Οι χρόνοι μπαίνουν σε ένα dictionary, το οποίο επιστρέφεται μαζί με True ή False για κάθε αποτέλεσμα. Το True ή False υπολογίζεται από τη συνάρτηση **ResultOK**, η οποία παίρνει το αποτέλεσμα των αλγορίθμων και ελέγχει αν όντως δεν παραβιάζονται οι

περιορισμοί μεταξύ των μεταβλητών με τις τιμές που τους έχουν ανατεθεί. Αν δεν υπάρχουν αναθέσεις τιμών τότε σημαίνει ότι έχει αποτύχει ο αλγόριθμος στο να βρει λύση με ικανοποίηση περιορισμών (UNSAT). Η συνάρτηση αυτή, επίσης, υπολογίζει πόσες αναθέσεις τιμών (assignments) έγιναν και πόσοι έλεγχοι περιορισμών (constraintchecks) έγιναν. Τα assignments υπολογίζονται από την ήδη υπάρχουσα υλοποίηση του CSP, ενώ τα constraint checks τα υπολογίζω εγώ χρησιμοποιώντας έναν μετρητή, τον οποίο αυξάνω κατά 1 κάθε φορά που καλείται η συνάρτηση constraints. Αυτά τα 2 στοιχεία εκτυπώνονται κατά την εκτέλεση της συνάρτησης.

Στο αρχείο **main.py**, καλείται η runFiles για κάθε τριάδα αρχείων με ίδιο αριθμό π.χ. var11, dom11, ctr11 και παίρνει τα αποτελέσματα και τα βάζει σε ένα dictionary το οποίο εκτυπώνεται στο τέλος. Τα αποτελέσματα είναι της μορφής: "filesXXXXX result [((mac, dom_wdeg), y), ((fc, dom_wdeg), y), ((fc-cbj, dom_wdeg), y)]" όπου XXXX = αριθμός του αρχείου, result = αποτέλεσμα αλγορίθμου (SAT/UNSAT) και y = χρόνος εκτέλεσης. Έχω αφήσει σε σχόλια τα αρχεία 8-f10 και 14-f27 γιατί δεν τελείωνε ποτέ το πρόγραμμα για όλους τους αλγορίθμους (για 8-f10 έχω αποτελέσματα με MAC και FC-CBJ, ενώ για 14-f27 έχω μόνο με FC-CBJ) κι έτσι τρέχω μόνο τα υπόλοιπα αρχεία που βγάζουν αποτελέσματα και για τους 3. Αν θέλετε μπορείτε να βγάλετε από τα σχόλια και τα 2 προαναφερθείσα αρχεία, αρκεί στο αρχείο findPairs.py, στη συνάρτηση runFiles να καλέσετε τους αλγορίθμους για τους οποίους βρήκα αποτελέσματα.

Πρόβλημα 2:

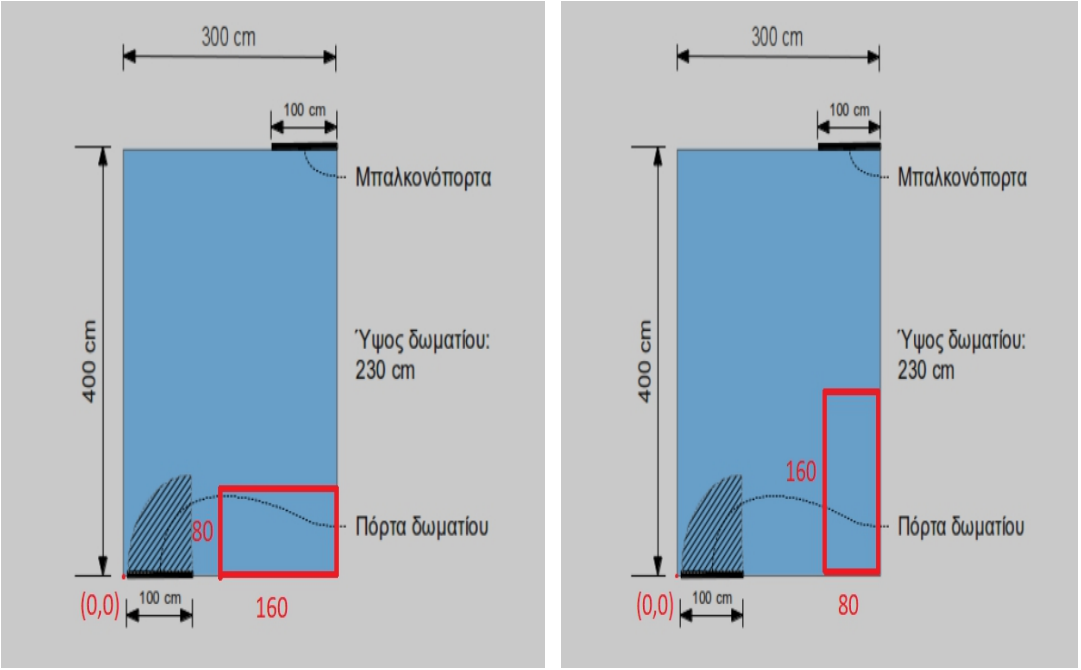
Σύμφωνα με τη θεωρία για να ορίσουμε ένα πρόβλημα ικανοποίησης περιορισμών, χρειαζόμαστε 3 βασικά πράγματα. Τις μεταβλητές και το πεδίο των δυνατών τιμών τους, καθώς και τους περιορισμούς τιμών που δεν πρέπει να παραβιάζουν κάποιες μεταβλητές μεταξύ τους.

Στο συγκεκριμένο πρόβλημα, έχουμε ένα φοιτητικό δωμάτιο, το οποίο φαίνεται στην εικόνα να έχει δύο διαστάσεις, αλλά στην πραγματικότητα έχει 3 διαστάσεις (Πλάτος, Μήκος / Βάθος, Ύψος), όπως και εξηγεί η εκφώνηση. Άρα το δωμάτιο μπορεί να αναπαρασταθεί με το Καρτεσιανό σύστημα συντεταγμένων, χρησιμοποιώντας 3 διαστάσεις. Τα σύστημα μέτρησης είναι σε εκατοστά (cm).

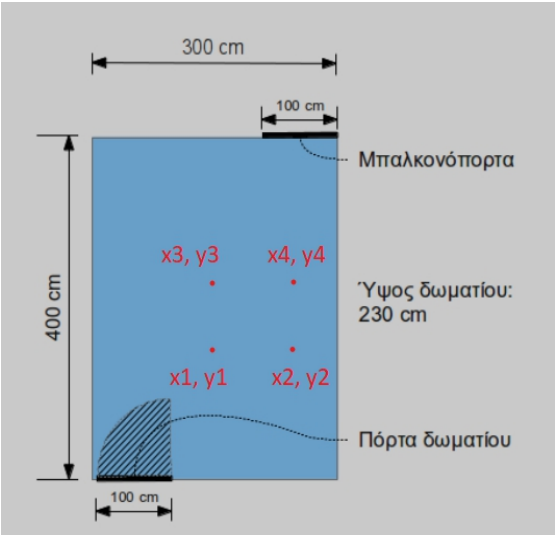
Οι **μεταβλητές** στο πρόβλημα μας είναι το **Κρεβάτι**, το **Γραφείο** και η **Καρέκλα** του και τέλος ο **Καναπές**.

Όσον αφορά το **πεδίο των δυνατών τιμών** για κάθε μεταβλητή, από την εικόνα και τις διαστάσεις που μας δίνονται, καταλαβαίνουμε ότι μιλάμε για παραλληλεπίπεδα σχήματα 3 διαστάσεων. Επειδή στην εικόνα έχουμε κάτοψη, το ύψος όλων των επίπλων είναι κατά πολύ μικρότερο από το ύψος του δωματίου και προφανώς δε θα τοποθετήσουμε το ένα πάνω στο άλλο, δε μας ενδιαφέρει το ύψος στους περιορισμούς. Οπότε για κάθε έπιπλο, θα διαλέγουμε 4 σημεία στο επίπεδο,

τα οποία αντιστοιχούν στις 4 άκρες(γωνίες) του επίπλου, θα ορίζουν ένα παραλληλεπίπεδο που θα οριοθετείται από τις ευθείες γραμμές που ενώνουν τα σημεία περιμετρικά και θα έχουν τις παρακάτω ιδιότητες. Οι ευθείες γραμμές που τα ενώνουν ανά ζεύγη των 2, οι οποίες είναι παράλληλες με τους άξονες x, y αποτελούν τις πλευρές των σχημάτων. Δηλαδή, ανάλογα με το πώς θα τοποθετηθεί το κάθε έπιπλο, οι 2 παράλληλες ευθείες που είναι παράλληλες με τον άξονα x αποτελούν το πλάτος ή το μήκος/βάθος των επίπλων και αντίστοιχα οι άλλες 2 παράλληλες ευθείες που είναι παράλληλες με τον άξονα y αποτελούν το μήκος/βάθος ή το πλάτος. Επίσης αυτά τα ζεύγη σημείων, όσον αφορά το πλάτος θα έχουν απόσταση dx ίση με το πλάτος του έπιπλου και όσον αφορά το μήκος/βάθος θα έχουν απόσταση dy ίση με το μήκος/βάθος. Οι δύο παραπάνω περιπτώσεις αναλύονται καλύτερα παρακάτω. Πιο συγκεκριμένα, για παράδειγμα, το Γραφείο με πλάτος 160cm και βάθος 80cm, θα μπορούσε να τοποθετηθούν οι γωνίες του στα 4 σημεία (140, 0, 90) (300, 0, 90) (140, 80, 90) (300, 80, 90), με σημείο τομής (αρχή) των αξόνων x, y, z το (0, 0, 0) το οποίο έστω ότι είναι η κάτω αριστερή γωνία του δωματίου, όπως φαίνεται στην εικόνα. Αντίστοιχα, με στροφή 90 μοιρών θα μπορούσε να τοποθετηθεί στα 4 σημεία (220, 0, 90) (300, 0 ,90) (220, 160, 90) (300, 160, 90). Η διαφορά φαίνεται καλύτερα στις εικόνες παρακάτω, αλλά σε επίπεδο 2 διαστάσεων.



Έστω ότι τα έπιπλα τοποθετούνται σε 4 σημεία με αυτήν τη σειρά (x1, y1, z) (x2, y2, z) (x3, y3, z) (x4, y4, z), όπως φαίνεται παρακάτω:

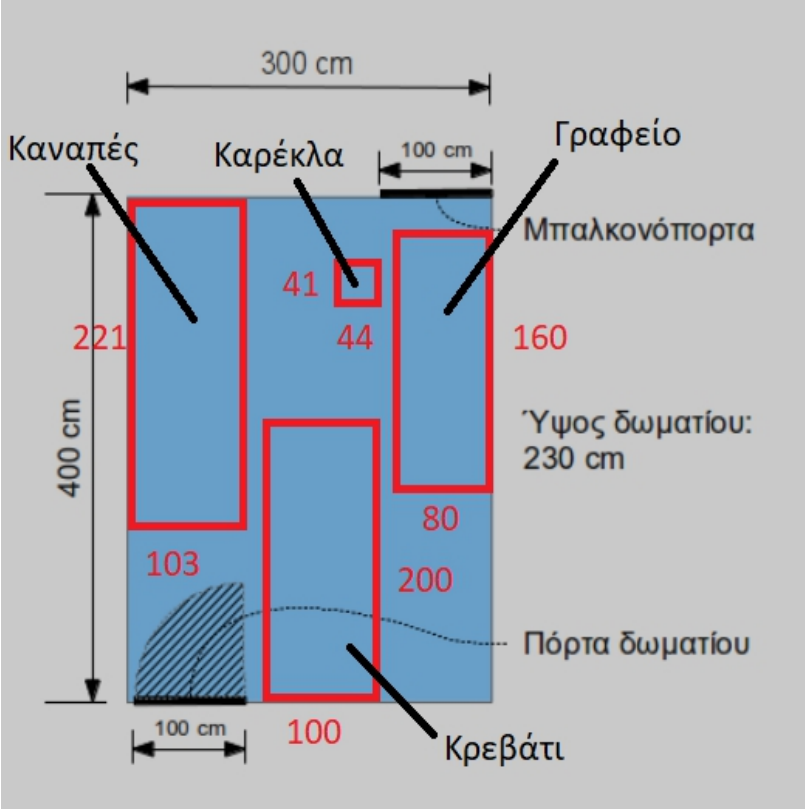


Άρα τα πεδία των δυνατών τιμών είναι (τα διαζευκτικά “ή” σημαίνουν ότι αλλάζουν οι πλευρές του επίπλου ανάλογα με το πως τοποθετείται, όπως φαίνεται στο ζεύγος των εικόνων παραπάνω): **Κρεβάτι** (x1, y1, 80) (x2, y2, 80) (x3, y3, 80) (x4, y4, 80) με x1, x2, x3, x4, y1, y2, y3, y4 τέτοια ώστε |x1 - x2| = 100 (πλάτος) ή 200 (μήκος), |x3 - x4| = 100 (πλάτος) ή 200 (μήκος), |y1 - y3| = 200 (μήκος) ή 100 (πλάτος), |y2 - y4| = 200 (μήκος) ή 100 (πλάτος) , **Γραφείο** (x1, y1, 90) (x2, y2, 90) (x3, y3, 90) (x4, y4, 90) με x1, x2, x3, x4, y1, y2, y3, y4 τέτοια ώστε |x1 - x2| = 160 (πλάτος) ή 80 (βάθος), |x3 - x4| = 160 (πλάτος) ή 80 (βάθος), |y1 - y3| = 80 (βάθος) ή 160 (πλάτος), |y2 - y4| = 80 (βάθος) ή 160 (πλάτος), **Καρέκλα Γραφείου** (x1, y1, 57) (x2, y2, 57) (x3, y3, 57) (x4, y4, 57) με x1, x2, x3, x4, y1, y2, y3, y4 τέτοια ώστε |x1 - x2| = 41 (πλάτος) ή 44 (βάθος), |x3 - x4| = 41 (πλάτος) ή 44 (βάθος), |y1 - y3| = 44 (βάθος) ή 41 (πλάτος), |y2 - y4| = 44 (βάθος) ή 41 (πλάτος) και ο **Καναπές** (x1, y1, 84) (x2, y2, 84) (x3, y3, 84) (x4, y4, 84) με x1, x2, x3, x4, y1, y2, y3, y4 τέτοια ώστε |x1 - x2| = 221 (πλάτος) ή 103 (βάθος), |x3 - x4| = 221 (πλάτος) ή 103 (βάθος), |y1 - y3| = 103 (βάθος) ή 221 (πλάτος), |y2 - y4| = 103 (βάθος) ή 221 (πλάτος).

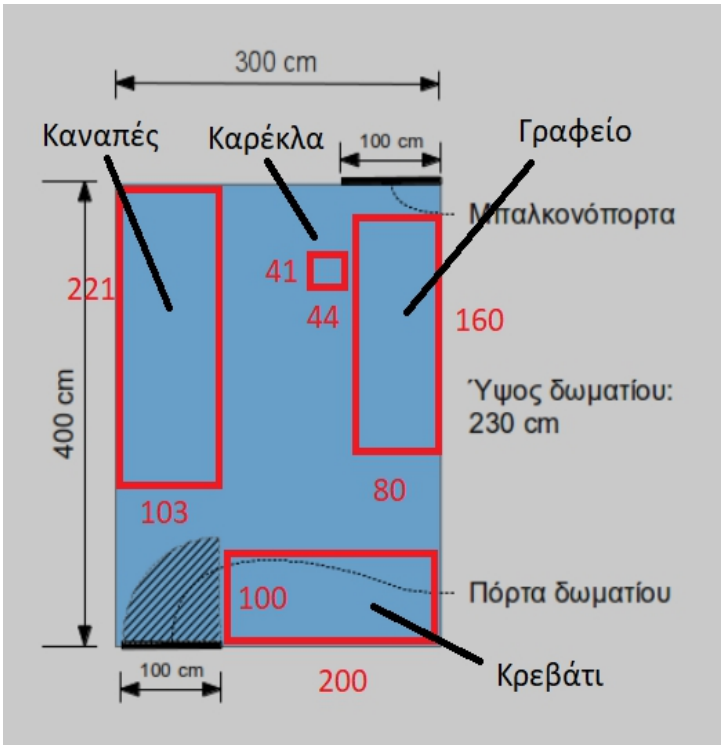
Όσον αφορά τους **περιορισμούς**, τα έπιπλα θα πρέπει να χωράνε στο δωμάτιο, δηλαδή τα σημεία που θα τοποθετηθούν να μην είναι εκτός του παραλληλεπίπεδου που σχηματίζουν οι 4 τοίχοι του δωματίου. Πιο συγκεκριμένα, με (0, 0, 0) σημείο τομής των αξόνων, τα έπιπλα πρέπει να τοποθετηθούν εντός του παραλληλεπιπέδου που ορίζουν τα σημεία (0, 0, 230) (300, 0, 230) (0, 400, 230) (300, 400, 230). Επίσης πρέπει να μην εφάπτονται μεταξύ τους και να μην είναι το ένα πάνω στο άλλο. Δηλαδή το κάθε έπιπλο καλύπτει κι αυτό ένα παραλληλεπίπεδο, το οποίο ορίζεται από τα 4 σημεία στα οποία τοποθετούνται οι γωνίες του, συνεπώς κανένα έπιπλο δεν επιτρέπεται καμία από τις 4 γωνίες του να τοποθετηθεί μέσα στο παραλληλεπίπεδο που ορίζεται από άλλα έπιπλα. Για να μην εφάπτονται, θα πρέπει επίσης τα σημεία των περιφερειών τους να έχουν απόσταση μεγαλύτερη του 1. Ένας ακόμη περιορισμός δημιουργείται λόγω της πόρτας, η οποία ανοίγει προς τα μέσα, σύμφωνα με την εικόνα, άρα εντός του τεταρτοκύκλιου πr^2 = 100 που ορίζεται από τα 3 σημεία (0, 0, y) (100, 0, y) (100, 100, y) με y = ύψος πόρτας, δεν μπορεί να τοποθετηθεί κάποιο έπιπλο. Τέλος το γραφείο, πρέπει να είναι δίπλα σε

είσοδο φωτός, άρα θα τοποθετηθεί κοντά στην μπαλκονόπορτα, αλλά όχι μπροστά της, έτσι ώστε να μπορεί κι η ίδια να χρησιμοποιείται.

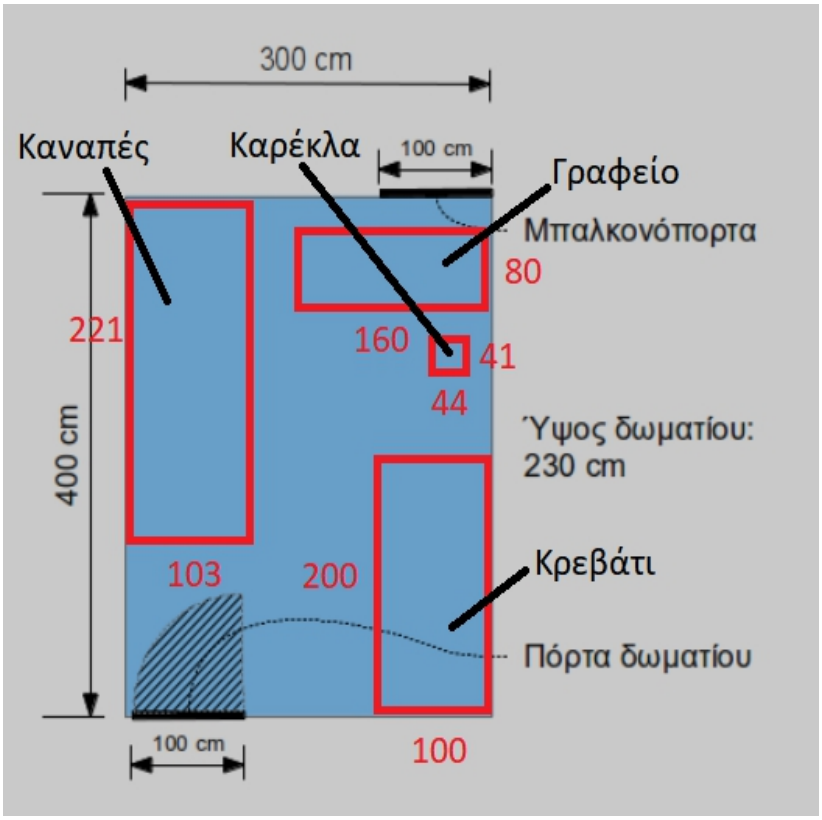
Μια λύση του προβλήματος φαίνεται στην παρακάτω εικόνα, αν και στην πραγματικότητα δε θα εφαρμοζόταν. Αυτό διότι, στενεύει ο χώρος, δεν υπάρχουν εύκολα περάσματα και δεν είναι αισθητικά ωραίο.



Μια καλύτερη λύση θα ήταν η παρακάτω, αλλά δεν μπορώ να καταλάβω με σιγουριά, αν το διάστημα μήκους 100 cm που καλύπτει η πόρτα αριστερά απ' το κρεβάτι, ξεκινά από τη γωνία ακριβώς. Αν το άνοιγμα της πόρτας ξεκινάει ακριβώς από την κάτω αριστερά γωνία του δωματίου, τότε το κρεβάτι χωράει οριακά (100 + 200) και η λύση παρακάτω είναι αποδεκτή. Αν ξεκινάει λίγο πιο μετά τη γωνία, τότε η μόνη αποδεκτή λύση είναι η προηγούμενη.



Μια ακόμα λύση θα μπορούσε να είναι η παρακάτω:



Πρόβλημα 3:

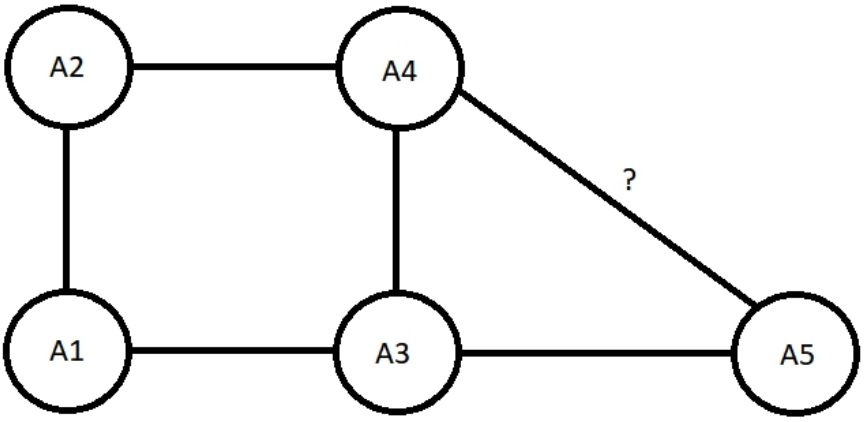
1) Θα ορίσουμε το πρόβλημα, όπως και στο προηγούμενο ερώτημα. Έχουμε να χρονοπρογραμματίσουμε 5 ενέργειες.

Οι **μεταβλητές** είναι οι 5 ενέργειες: A1, A2, A3, A4, A5.

Τα **πεδία των δυνατών τιμών** των μεταβλητών είναι, για κάθε ενέργεια, το χρονικό διάστημα κατά το οποίο θα εκτελείται. Όλες οι ενέργειες μπορούν να ξεκινήσουν είτε στις 9:00 είτε στις 10:00 είτε στις 11:00. Κάθε ενέργεια διαρκεί 60 λεπτά. Άρα οι δυνατές τιμές είναι: 9:00 - 10:00, 10:00 - 11:00, 11:00 - 12:00.

Οι **περιορισμοί** είναι αυτοί που αναφέρονται στην εκφώνηση. Η A1 πρέπει να αρχίσει μετά την A3. Η A3 πρέπει να αρχίσει πριν την A4 και μετά την A5. Η A2 δεν μπορεί να εκτελείται την ίδια ώρα με την A1 ή την A4. Η A4 δεν μπορεί να αρχίσει στις 10:00. Άρα **Start(A1) > Start(A3)**, **Start(A5) < Start(A3) < Start(A4)**, **Start(A4) != 10:00** , **RunningTime(A2) != RunningTime(A1) or RunningTime(A4).**

2)



Το ερωτηματικό δίπλα από την ακμή που ενώνει τις ενέργειες A4 και A5 το έβαλα επειδή, από τη μία προκύπτει περιορισμός μεταξύ τους λόγω της πρότασης “Η A3 πρέπει να αρχίσει πριν την A4 και μετά την A5”, που σημαίνει ότι εφ’ όσον η A3 πρέπει να αρχίσει πριν την A4 και μετά την A5, η A5 θα πρέπει να αρχίσει πριν την A3, η οποία θα πρέπει να αρχίσει πριν την A4, συνεπώς η A5 θα πρέπει να αρχίσει πριν την A4. Από την άλλη, δε δηλώνεται ρητά στην εκφώνηση ο περιορισμός μεταξύ των δύο αυτών ενεργειών. Κατά τη γνώμη μου, ο περιορισμός μεταξύ A5 και A4 υπάρχει σίγουρα.

3)

Αλγόριθμος AC-3

Από τον γράφο των περιορισμών, παίρνουμε τις ακμές που ενώνουν τις ενέργειες και τις βάζουμε στην ατζέντα κι από τις 2 κατευθύνσεις. Έχουμε: A1 > A3, A3 < A1, A1 != A2, A2 != A1, A2 != A4, A4 != A2, A4 > A3, A3 < A4, A5 < A3, A3 > A5, A5 < A4, A4 > A5.

Agenda = { A1-A3, A3-A1, A1-A2, A2-A1, A2-A4, A4-A2, A4-A3, A3-A4, A5-A3, A3-A5, A5-A4, A4-A5 }

Dom(A1) = { 9, 10, 11 }
Dom(A2) = { 9, 10, 11 }
Dom(A3) = { 9, 10, 11 }
Dom(A4) = { 9, 11 } (Η A4 δεν μπορεί να αρχίσει στις 10, σύμφωνα με την εκφώνηση)
Dom(A5) = { 9, 10, 11 }

Αρχικά αφαιρούμε από την ατζέντα την ακμή **A1-A3** και κοιτάμε τις τιμές της A1. Αν η A1 ξεκινήσει στις 9, καμία τιμή της A3 δεν ικανοποιεί τον περιορισμό άρα αφαιρούμε από το domain της A1 την τιμή 9. Άμα πάρει τιμή 10 η A1, τότε υπάρχουν διαθέσιμες τιμές για την A3. Το ίδιο ισχύει, αν πάρει τιμή 11 η A1. Επειδή άλλαξε το domain της A1 πρέπει να βάλουμε στην ατζέντα όσες ακμές έχουν στο δεξί άκρο τους την A1. Αφού όμως, είμαστε ακόμα στην αρχή του αλγορίθμου, δε χρειάζεται να προσθέσουμε κάποια, εφόσον υπάρχουν όλες ήδη μέσα.

Τώρα έχουμε:

Dom(A1) = { 10, 11 }
Dom(A2) = { 9, 10, 11 }
Dom(A3) = { 9, 10, 11 }
Dom(A4) = { 9, 11 }
Dom(A5) = { 9, 10, 11 }
Agenda = { A3-A1, A1-A2, A2-A1, A2-A4, A4-A2, A4-A3, A3-A4, A5-A3, A3-A5, A5-A4, A4-A5 }

Θα επαναλάβουμε την παραπάνω διαδικασία, μέχρις ότου αδειάσει η ατζέντα.

Αφαιρούμε την ακμή **A3-A1**. Για A3: άμα πάρει τιμή 9 ή 10 δεν υπάρχει παραβίαση περιορισμού, όμως άμα πάρει τιμή 11 τότε δεν υπάρχουν διαθέσιμες επιτρεπτές τιμές για την A1. Οπότε αφαιρούμε την τιμή 11 από το domain της A3. Αφού άλλαξε το domain της A3, βάζουμε στο τέλος της ατζέντας την A1-A3.

Τώρα έχουμε:

Dom(A1) = { 10, 11 }
Dom(A2) = { 9, 10, 11 }
Dom(A3) = { 9, 10 }
Dom(A4) = { 9, 11 }
Dom(A5) = { 9, 10, 11 }

Agenda = { A1-A2, A2-A1, A2-A4, A4-A2, A4-A3, A3-A4, A5-A3, A3-A5, A5-A4, A4-A5, A1-A3 }

Αφαιρούμε την ακμή **A1-A2**. Για A1: με τιμή 10 υπάρχουν τιμές για την A2, το ίδιο και με τιμή 11. Εδώ δεν άλλαξε κάτι, άρα προχωράμε κατευθείαν στην επόμενη ακμή.

Αφαιρούμε την ακμή **A2-A1.** Για όλες τις τιμές της A2, υπάρχουν επιτρεπτές τιμές για την A1, οπότε προχωράμε.

Dom(A1) = { 10, 11 }

Dom(A2) = { 9, 10, 11 }

Dom(A3) = { 9, 10 }

Dom(A4) = { 9, 11 }

Dom(A5) = { 9, 10, 11 }

Agenda = { A2-A4, A4-A2, A4-A3, A3-A4, A5-A3, A3-A5, A5-A4, A4-A5, A1-A3 }

Αφαιρούμε την ακμή **A2-A4.** Για όλες τις τιμές της A2, υπάρχουν επιτρεπτές τιμές για την A4, οπότε προχωράμε.

Αφαιρούμε την ακμή **A4-A2**. Για όλες τις τιμές της A4, υπάρχουν επιτρεπτές τιμές για την A2, οπότε προχωράμε.

Agenda = { A4-A3, A3-A4, A5-A3, A3-A5, A5-A4, A4-A5, A1-A3 }

Αφαιρούμε την ακμή **A4-A3**. Για A4: άμα πάρει τιμή 11 όλα είναι εντάξει αλλά άμα πάρει 9 δεν υπάρχει τιμή για την A3 που να ικανοποιεί τον περιορισμό, άρα αφαιρούμε την τιμή 9 από την A4. Πρέπει να βάλουμε και τις ακμές με δεξί άκρο την A4, δηλαδή την A2-A4 διότι οι άλλες υπάρχουν ήδη.

Τώρα έχουμε:

Dom(A1) = { 10, 11 }

Dom(A2) = { 9, 10, 11 }

Dom(A3) = { 9, 10 }

Dom(A4) = { 11 }

Dom(A5) = { 9, 10, 11 }

Agenda = { A3-A4, A5-A3, A3-A5, A5-A4, A4-A5, A1-A3, A2-A4 }

Αφαιρούμε την **A3-A4**. Για όλες τις τιμές της A3, υπάρχουν διαθέσιμες επιτρεπτές τιμές για την A4, οπότε προχωράμε.

Αφαιρούμε την **A5-A3**. Η τιμή 9 της A5 είναι εντάξει, όμως για 10 και 11 δεν υπάρχουν διαθέσιμες τιμές για την A3, άρα τις αφαιρούμε. Όλες οι ακμές με δεξί άκρο την A5 είναι ήδη μέσα στην ατζέντα.

Dom(A1) = { 10, 11 }

Dom(A2) = { 9, 10, 11 }

Dom(A3) = { 9, 10 }

Dom(A4) = { 11 }

Dom(A5) = { 9 }

Agenda = { A3-A5, A5-A4, A4-A5, A1-A3, A2-A4 }

Αφαιρούμε την **A3-A5**. Αφαιρούμε την τιμή 9 από το domain της A3. Προσθέτουμε στο τέλος της ατζέντας τις ακμές με δεξί άκρο A3: A4-A3, A5-A3. Η A1-A3 υπάρχει ήδη μέσα.

Dom(A1) = { 10, 11 }

Dom(A2) = { 9, 10, 11 }

Dom(A3) = { 10 }

Dom(A4) = { 11 }

Dom(A5) = { 9 }

Agenda = { A5-A4, A4-A5, A1-A3, A2-A4, A4-A3, A5-A3 }

Αφαιρούμε την **A5-A4.** Οι τιμές είναι εντάξει και δεν αλλάζει τίποτα.

Αφαιρούμε την **A4-A5**. Οι τιμές είναι εντάξει και δεν αλλάζει τίποτα.

Αφαιρούμε την **A1-A3**. Η τιμή 10 δημιουργεί πρόβλημα και την αφαιρούμε από το domain της A1, η 11 είναι εντάξει. Βάζουμε στο τέλος της ατζέντας τις ακμές A3-A1, A2-A1.

Dom(A1) = { 11 }

Dom(A2) = { 9, 10, 11 }

Dom(A3) = { 10 }

Dom(A4) = { 11 }

Dom(A5) = { 9 }

Agenda = { A2-A4, A4-A3, A5-A3, A3-A1, A2-A1 }

Αφαιρούμε την **A2-A4**. Η τιμές 9 και 10 είναι εντάξει, η 11 όμως δημιουργεί πρόβλημα οπότε την αφαιρούμε από το domain της A2. Βάζουμε στο τέλος της ατζέντας τις ακμές A1-A2, A4-A2.

Dom(A1) = { 11 }

Dom(A2) = { 9, 10 }

Dom(A3) = { 10 }

Dom(A4) = { 11 }

Dom(A5) = { 9 }

Agenda = { A4-A3, A5-A3, A3-A1, A2-A1, A1-A2, A4-A2 }

Αφαιρούμε την **A4-A3**. Οι τιμές είναι εντάξει και δεν αλλάζει τίποτα.

Αφαιρούμε την **A5-A3**. Οι τιμές είναι εντάξει και δεν αλλάζει τίποτα.

Αφαιρούμε την **A3-A1**. Οι τιμές είναι εντάξει και δεν αλλάζει τίποτα.

Αφαιρούμε την **A2-A1**. Οι τιμές είναι εντάξει και δεν αλλάζει τίποτα.

Αφαιρούμε την **A1-A2**. Οι τιμές είναι εντάξει και δεν αλλάζει τίποτα.

Αφαιρούμε την **A4-A2**. Οι τιμές είναι εντάξει και δεν αλλάζει τίποτα.

Η ατζέντα άδειασε, άρα εδώ τελειώνει ο αλγόριθμος.

Τα τελικά domains είναι τα εξής:

Dom(A1) = { 11 }

Dom(A2) = { 9, 10 }

Dom(A3) = { 10 }

Dom(A4) = { 11 }

Dom(A5) = { 9 }