Τεχνητή Νοημοσύνη ~ Εργασία 3

Μυλωνόπουλος Δημήτριος – Α.Μ: 1115201600112

Πρόβλημα 2:

Για να επιλύσουμε το δωθέν πρόβλημα, το πρόβλημα μοντελοποιήθηκε σε πρόβλημα ικανοποίησης περιορισμών CSP, με τρόπο τέτοιο ώστε να δωθεί λύση στο πρόβλημα τοποθέτησης των επιπλών στο δωμάτιων τηρώντας πάντα τους κανόνες αισθητικής που έχουν ζητηθεί.

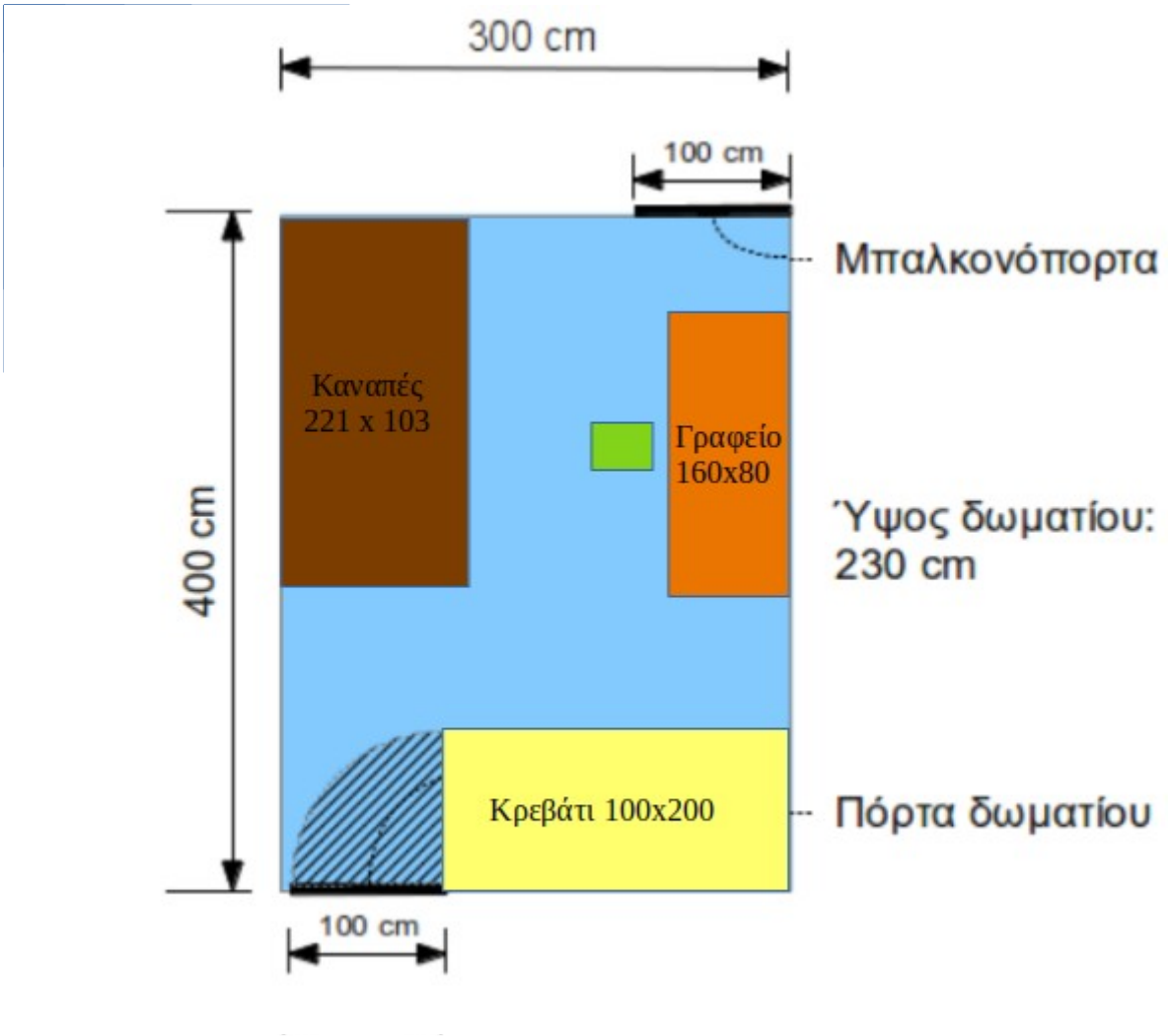
Μεταβλητές: Ορίζω ως Μεταβλητές X1,X2 κλπ τις συντεταγμένες της γωνίας του επίπλου εστω (i,j) που έχουν την ελάχιστη τετμημένη και τεταγμένη απ' τις υπόλοιπες του γωνιές και το ονομα του επίπλου (κατω αριστερά γωνία), όπως και το όνομα του επίπλου σε ενα ζευγάρι ((i,j), όνομα_επίπλου). (Τις υπόλοιπες γωνίες της βρίσκουμε εύκολα γνωρίζοντας τις διαστάσεις του επίπλου δεδομένουν ότι τα έπιπλα δεν περιστρέφονται στον χώρο).

Πεδίο τιμών: Για ένα δωμάτιο διαστάσεων nxm το πεδίο τιμών μιας μεταβλητής θα είναι για το πρώτο κομμάτι της δυάδας, το σύνολο των συντεταγμένων του χώρου δηλαδή τα (i,j) με 1<=i<=300 και 1<=j<=400, και για το δευτερο τα διαθέσιμα ονοματα των επίπλων (Καναπες, Κρεβάτι, Καρέκλα, Γραφείο).

Περιορισμοί:

- Εστώ η τιμή μιας μεταβλητής X, Y δυο μεταβλητές με τιμές (i_1,j_1) και (i_2, j_2): i_1 + ΜΗΚΟΣ(X) >= i_2 >= i_1 ή i_1 + ΜΗΚΟΣ(X) >= i_2 + ΜΗΚΟΣ(Y) >= i_1 => j_1 > j_2 + ΠΛΑΤΟΣ(Y) ή j_2 > j_1 + ΠΛΑΤΟΣ(X)
- Εστώ η τιμή μιας μεταβλητής X, Y δυο μεταβλητές με τιμές (i_1,j_1) και (i_2, j_2):
- j_1 + ΠΛΑΤΟΣ(X) >= j_2 >= j_1 ή j_1 + ΠΛΑΤΟΣ(X) >= j_2 + ΠΛΑΤΟΣ(Y) >= j_1=> i_1 + ΜΗΚΟΣ(X) < i_2 ή i_1 > i_2 + ΜΗΚΟΣ(Y)
- (προαιρετικός περιορισμός)Ακόμη για να μπορέσει ένα έπιπλο να χρησιμοποιηθεί θα πρέπει το υψος του να είναι μικρότερο απ' το ύψος του χώρου στο οποίο θα τοποθετηθεί.
- Επιπλέον για να είναι το γραφείο διπλα σε καποια πηγή φωτός θα πρέπει να εφάπτεται στον δεξια τοίχο , άρα θα πρέπει για τη μεταβλητή του γραφείου X με συντεταγμένες (i, j) να ισχύει i > 200.
- Επιπλέον για να είναι έγκυρη η θέση του επίπλου θα πρέπει για μεταβλητή X με συντεταγμένες (i,j) να α ισχύει: i + ΜΗΚΟΣ < 300 και j+ ΠΛΑΤΟΣ < 400. Αυτό αν θεωρήσουμε πως η κάτω αριστερά γωνία έχει συντεταγμένες (1,1)
- Ακόμα για να μην χτυπάει το έπιπλο στην πόρτα θα πρέπει: Έστω μεταβλητή X με συντεταγμένες (i,j) i > 100 και j >100.

Μια λύση για το δωθέν πρόβλημα βασισμένη στους παραπάνω περιορισμους και μοντελοποιηση είναι η εξής:


300 cm
100 cm
Μπαλκονόπορτα
Καναπές
221 x 103
Γραφείο
160x80
400 cm
Ύψος δωματίου:
230 cm
Κρεβάτι 100x200
Πόρτα δωματίου
100 cm

Πρόβλημα 3:

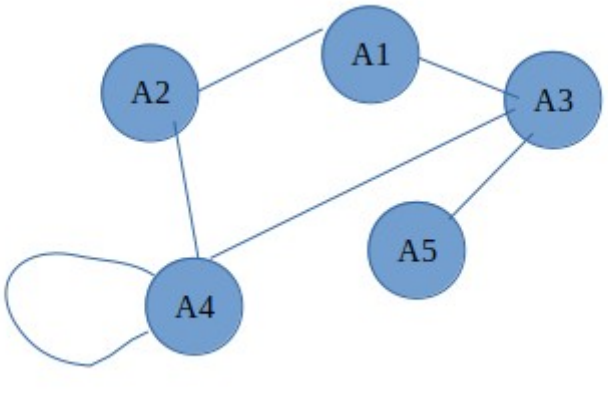
Μεταβλητές: A1...A5

Πεδίο τιμών:Για όλες τις μεταβλητές είναι **{**9:00, 10:00, 11:00**}** εκτός απο την A4 που είναι {9:00,11:00}

Περιορισμοί:

- **A1 > A3**
- **A3< A4**
- **A5<A3**
- **A2 <> A1**
- **A2<>A4**

2.



3.

Αρχικά Domains:
D(A1) = { 9:00, 10:00, 11:00}
D(A2) = { 9:00, 10:00, 11:00}
D(A3) = { 9:00, 10:00, 11:00}
D(A4) = { 9:00, 11:00} //Δεν μπορεί να εκτελεστεί στις 10:00 λόγω unary restriction από εκφώνηση
D(A5) = { 9:00, 10:00, 11:00}

Queue = Tail → {(A1, A2), (A2,A1), (A1, A3), (A3, A1), (A2, A4), (A4, A2), (A4, A3), (A3, A4), (A3, A5), (A5, A3)} ← Head

pop → (A5,A3)

Remove Incons(A5,A3)
vA5 = 11:00 unsatisfied επειδή πρέπει A5 < A3 και δεν γινεται για την τιμη 11:00
---D(A5) = {9:00, 10:00}

Queue = Tail → {(A3, A5), (A1, A2), (A2,A1), (A1, A3), (A3, A1), (A2, A4), (A4, A2), (A4, A3), (A3, A4), (A3, A5))} ← Head

---

pop → (A3, A5)

Remove Incons(A3, A5)
vA3 = 9:00 unsatisfieds, επειδή πρέπει A3>A5 και δεν γινεται για την τιμη 9:00
--D(A3) = {10:00, 11:00}

Queue = Tail → {(A5,A3), (A4,A3), (A1,A3),(A3, A5), (A1, A2), (A2,A1), (A1, A3), (A3, A1), (A2, A4), (A4, A2), (A4, A3), (A3, A4)} ← Head

---

pop →(A3,A4)
Remove Incons(A3,A4)
vA3 = 11:00 unsatisfied, επειδή A3<A4 και αυτό δεν γίνεται για την τιμή 11:00
--D(A3) = {10:00}

Queue = Tail → {(A5,A3), (A4,A3), (A1,A3),(A5,A3), (A4,A3), (A1,A3),(A3, A5), (A1, A2), (A2,A1), (A1, A3), (A3, A1), (A2, A4), (A4, A2), (A4, A3)} ← Head

---

pop → (A4, A3)

Remove Incons(A4, A3)
vA4 = 9:00 unsatisfied, επειδή A4>A3 και αυτό δεν γινεται για την τιμή 9:00
--D(A4) = {11:00}

Queue = Tail → {(A2, A4), (A3, A4), (A5,A3), (A4,A3), (A1,A3),(A5,A3), (A4,A3), (A1,A3),(A3, A5), (A1, A2), (A2,A1), (A1, A3), (A3, A1), (A2, A4), (A4, A2)} ← Head

---

pop → (A4, A2)

Remove Incons (A4, A2) return false

Queue = Tail → {(A2, A4), (A3, A4), (A5,A3), (A4,A3), (A1,A3),(A5,A3), (A4,A3), (A1,A3),(A3, A5), (A1, A2), (A2,A1), (A1, A3), (A3, A1), (A2, A4)} ← Head

---

pop → (A2, A4)

RemoveIncons(A2,A4)
vA2 = 11:00 unsatisfied, επειδή το A4 έχει μονο την τιμή 11:00 και A2 <> A4
----D(A2) = {9:00, 10:00}

Queue = Tail → {(A1,A2), (A4,A2),(A2, A4), (A3, A4), (A5,A3), (A4,A3), (A1,A3),(A5,A3), (A4,A3), (A1,A3),(A3, A5), (A1, A2), (A2,A1), (A1, A3), (A3, A1)} ← Head

---

pop → (A3, A1)

Remove Incons(A3,A1) return false

Queue = Tail → {(A1,A2), (A4,A2),(A2, A4), (A3, A4), (A5,A3), (A4,A3), (A1,A3),(A5,A3), (A4,A3), (A1,A3),(A3, A5), (A1, A2), (A2,A1), (A1, A3)} ← Head

---

pop → (A1,A3)

RemoveIncons (A1, A3)
vA1 = {9:00, 10:00} unsatisfied, επειδή A1> A3 και A3 παίρνει μονο την τιμή 10:00
----D(A1) = {11:00}

Queue = Tail → {(A2,A1), (A3,A1),(A1,A2), (A4,A2),(A2, A4), (A3, A4), (A5,A3), (A4,A3), (A1,A3),(A5,A3), (A4,A3), (A1,A3),(A3, A5), (A1, A2), (A2,A1)} ← Head

---

pop → (A2, A1)

Remove Inconcs(A2, A1) return False

Queue = Tail → {(A2,A1), (A3,A1),(A1,A2), (A4,A2),(A2, A4), (A3, A4), (A5,A3), (A4,A3), (A1,A3),(A5,A3), (A4,A3), (A1,A3),(A3, A5), (A1, A2)} ← Head

---

pop → (A1, A2)

Remove Incons (A1, A2) return False

Queue = Tail → {(A2,A1), (A3,A1),(A1,A2), (A4,A2),(A2, A4), (A3, A4), (A5,A3), (A4,A3), (A1,A3),(A5,A3), (A4,A3), (A1,A3),(A3, A5)} ← Head

---

pop → (A3, A5)
Remove Incons(A3, A5) return False

Queue = Tail → {(A2,A1), (A3,A1),(A1,A2), (A4,A2),(A2, A4), (A3, A4), (A5,A3), (A4,A3), (A1,A3),(A5,A3), (A4,A3), (A1,A3)} ← Head

---

pop → (A1, A3)

Remove Incons(A1, A3) return False

Queue = Tail → {(A2,A1), (A3,A1),(A1,A2), (A4,A2),(A2, A4), (A3, A4), (A5,A3), (A4,A3), (A1,A3),(A5,A3), (A4,A3)} ← Head

---

pop → (A4, A3)

Remove Incons(A4, A3) return False

Queue = Tail → {(A2,A1), (A3,A1),(A1,A2), (A4,A2),(A2, A4), (A3, A4), (A5,A3), (A4,A3), (A1,A3),(A5,A3)} ← Head

---

pop → (A5, A3)

Remove Incons(A5, A3)
vA5 = 10:00 unsatisfied, επειδή A5 < A3 και A3 παίρνει μονο την τιμή 10:00
--D(A5) = {9:00}

Queue = Tail → {(A3, A5), (A2,A1), (A3,A1),(A1,A2), (A4,A2),(A2, A4), (A3, A4), (A5,A3), (A4,A3), (A1,A3)} ← Head

---

pop → (A1, A3)

RemoveIncons(A1, A3) return False

Queue = Tail → {(A3, A5), (A2,A1), (A3,A1),(A1,A2), (A4,A2),(A2, A4), (A3, A4), (A5,A3), (A4,A3)} ← Head

---

pop → (A4, A3)

Remove Incons(A4, A3) return False

Queue = Tail → {(A3, A5), (A2,A1), (A3,A1),(A1,A2), (A4,A2),(A2, A4), (A3, A4), (A5,A3)} ← Head

---

pop → (A5, A3)

Remove Incons(A5, A3) return False
Queue = Tail → {(A3, A5), (A2,A1), (A3,A1),(A1,A2), (A4,A2),(A2, A4), (A3, A4)} ← Head

---

pop → (A3, A4)

Remove Incons(A3, A4) return False

Queue = Tail → {(A3, A5), (A2,A1), (A3,A1),(A1,A2), (A4,A2),(A2, A4)} ← Head

---

pop → (A2, A4)

Remove Incons(A2, A4) return False

Queue = Tail → {(A3, A5), (A2,A1), (A3,A1),(A1,A2), (A4,A2)} ← Head

---

pop → (A4, A2)

Remove Incons(A4, A2) return False

Queue = Tail → {(A3, A5), (A2,A1), (A3,A1),(A1,A2)} ← Head

---

pop → (A1, A2)

Remove Incons(A1, A2) return False

Queue = Tail → {(A3, A5), (A2,A1), (A3,A1)} ← Head

---

pop → (A3, A1)

Remove Incons(A3, A1) return False

Queue = Tail → {(A3, A5), (A2,A1)} ← Head

---

pop → (A2, A1)

Remove Incons(A2, A1) return False

Queue = Tail → {(A3, A5)} ← Head

---

pop → (A3, A5)

Remove Incons(A3, A5) return False

Queue = Tail → {} ← Head

New Domains:

A1 = {11:00}
A2 = {9:00, 10:00}
A3 = {10:00}
A4 = {11:00}
A5 = {9:00}

Πρόβλημα 1(kenken puzzle):

1. Μοντελοποίηση Προβλήματος kenken σε binary CSP.

Για την μοντελοποίηση του προβλήματος KenKen σε binary CSP χρησιμοποιήθηκε η μέθοδος CSP για binarization, Dual Transformation. Ως ordinary variables του προβλήματος μπορούν να θεωρηθούν τα κελιά του KenKen puzzle. Εμείς ουσιαστικά για την μοντελοποίηση του προβλήματος θα πάρουμε τους περιορισμούς του αρχικού προβλήματος που ορίζουν μια κλίκα,

δηλαδή το σύνολο των στοιχείων- κελιών που ανήκουν σε μια κλίκα να υπακούν τους περιορισμούς που ορίζει η κλίκα. Πχ για πρόσθεση θέλουν το άθροισμα των τιμών των στοιχείων να είναι όσο ορίζεται από το πρόβλημα για τη συγκεκριμένη κλίκα. Εκτός τούτου άλλος ένα περιορισμός που περιλαμβάνεται στην dual variable είναι να μην έχουν τα στοιχεία του grid άρα και της κλίκας ίδια τιμή στην ίδια γραμμή ή στήλη με κάποιο άλλο στοιχείο που έχει ίδια τιμη. Άρα τελικά έχουμε την νέα dual variable κλίκα που περιλαμβάνει τους 2 παραπάνω περιορισμούς που ουσιαστικά ορίζουν την τιμή μιας κλίκα ως έγκυρη, χωρίς να λαμβάνουν υπόψιν τις υπόλοιπες κλικες (Κάτι που θα προσδιορίσουμε στη συνέχεια στα constraints του προβλήματος). Άρα θα έχουμε

Μεταβλητές: c1,c2,…,cn όπου ci μια κλίκα του προβλήματος kenken

Πεδίο τιμών: Το πεδίο τιμων για μια κλίκα είναι ενα σύνολο από tuples (a1,a2...ak) δηλαδή tuple μεγέθους k όσο δηλαδή και ο αριθμός των στοιχείων της κλίκας αυτής. Κάθε στοιχείο του tuple αντιπροσωπεύει την τιμή ενός κελιού της κλίκας και παίρνει τιμές από 1 έως n για ένα kenken puzzle μεγέθους nxn. Επίσης η τιμή της tuple πρέπει να είναι αποδεκτή σύμφωνα με τα constraints που έχουμε συμπεριλάβει στα dual variables.

Γείτονες: Ως γείτονες ορίζουμε δύο μεταβλητές A, B οι οποίες έχουν στοιχεία που ανήκουν στην ίδια γραμμή ή στήλη.

Περιορισμοί: Έστω δύο μεταβλητές A,B με τιμες a=(a1,a2...ai),b = (b1,b2...bj) αντίστοιχα τότε θα πρέπει για κάθε κελί xn με συντεταγμένες (n1,n2) της A και για κελί xm (m1,m2) της B:
n1 == m1 ή n2==m2 => τιμή xn <> τιμή xm

2. Για την υλοποίηση του προβλήματος σε python βασίστηκα στην υλοποίηση της σελίδας https://github.com/aimacode/aima-python/blob/master/csp.py για το πρόβλημα CSP. Όρισα την κλάση **kenken** που έχει ως πατρική την CSP που υλοποιείται στο csp.py. Στην κλάση μου ουσιαστικά δημιουργώ τις μεταβλητές, το πεδίο τιμών, τους γείτονες και τα constrains για το πρόβλημα του KenKen puzzle με την μοντελοποίηση που αναλύθηκε παραπάνω. Οι χάρτες του προβλήματος έχουν ορισθεί σε συναρτήσεις (**easy_map(), exerc_map(),hard() και expert()**) όπου και έχουν περαστεί “hard-coded”. Οι συναρτήσεις αυτές επιστρέφουν μια tuple που περιέχει το μέγεθος του n του χάρτη nxn, μια tuple με τις μεταβητές-κλίκες και ένα λεξικό που περιέχει την πράξη και τον αριθμό στόχο για την κάθε μεταβλητή-κλίκα.

Για την δημιουργία του domain εργάστηκα ως εξής. Εμείς θέλουμε να δημιουργήσουμε όλες τις δυνατές k άδες τιμών για μια μεταβλητή με αριθμό κελιών k, οι οποίες τιμές ικανοποιούν τους περιορισμούς που εχουμε ορίσει παραπάνω για μια κλίκα. Για να το κάνω αυτό χρησιμοποίησα μια βοηθήτική συνάρτηση την **AllValues** η οποία παίρνει ως όρισμα μια λίστα την **result,** το range των τιμών **n,** και το μέγεθος των παραγόμενων tuple μεγέθους k. Το **cage** που περιέχει την μεταβλητή για την οποία θέλουμε να παραξουμε τις τιμές και αντίστοιχο dictionary entry για την πράξη που θα χρησιμοποιηθεί και την τιμή στόχο για τη συγκεκριμένη μεταβλητή. Στην συνάρτηση δημιουργείτε μια λίστα με ακεραίους 1-n (numset) και καλείται η αναδρομική **AllValuesRec** η οποία έχει τα ίδια ορίσματα με την **AllValues** όμως περιλαμβάνει και το numset και το mylist που αρχικοποιείται με []. Η συνάρτηση αυτή δημιουργεί αναδρομικά όλες τις δυνατές κ-αδες με αυτό το πεδίο τιμών ανδρομικά. Στο βήμα βάσης δηλαδή για k=0, που σημαίνει οτι έχουμε φτιάξει μια λίστα τιμών μεγέθους k, ουσιαστικά ελέγχουμε αν η τιμή που έχουμε βρεί ειναι εγκυρη για τη συγκεκριμένη μεταβλητή. Δηλαδή αρχικά ελέγχουμε αν υπάρχουν στοιχεία με την ίδια τιμή στην ίδια γραμμή ή στηλη και αν υπάρχουν τοτε η τιμή δεν προσαρτάται στη λίστα result. Αν δεν υπάρχει τέτοιο θέμα τότε ανάλογα με το όρισμα **operation** που έχουμε ορίσει για την εξεταζόμενη μεταβλητή αποφασίζουμε αν θα κρατήσουμε την κ-αδα με κριτήριο το αν με την δωθείσα πράξη δίνει την τιμή στόχο για την συγκεκριμένη μεταβλητή. Αν δεν τη δίνει την απορρίπτουμε αλλιώς την προσθέτουμε

στην λίστα result. Άρα τελικα η result κρατάει όλες τις δυνατές tuples που μπορεί να πάρει η μεταβλητή μας. Την λίστα αυτή την περνούμε στο λεξικό domain για τη συγκεκριμένη μεταβλητή.

Για να δημιουργήσουμε το λεξικό με τους Neighbors ουσιαστικά για καθε κλειδί-μεταβλητή καταχωρούμε τις μεταβλητές με τις οποίες η μεταβλητή-κλειδί μοιράζεται κάποια γραμμή η στήλη του προβλήματος kenken.

Μόλις ορίσουμε τα πεδια Variables, Domains, Neighbors καλούμε την constructor της πατρικής κλάσης csp με ορίσματα τα παραπάνω πεδία όπως και την Kenken_Constraints που θα εξηγήσουμε παρακάτω.

Το Constraint function (**Kenken_Constraints)** παίρνει ως όρισμα 2 μεταβλητές Α,Β με τιμες α,β. Η συνάρτηση ελέγχει αν κάθε κελί της μεταβλητής Α δεν συγκρούεται με κάποιο κελί της μεταβλητής Β. Λέγοντας συγκρούεται εννοούμε αν ενα κελί της έχει την ίδια γραμμή ή στήλη με ένα κελί της Β να έχει και την ίδια τιμή. Σε αυτή την περίπτωση επιστρέφουμε False. Αν δεν υπάρξει “σύγκρουση” επιστρέφουμε True.

Επιπλέον υπάρχει μια συνάρτηση display η οποία εκτυπώνει το puzzle kenken. Αρχικά εκτυπώνονται οι κλικες. Αυτές αναπαρίστανται ως ενας πίνακας που σε καθε κελί μιας κλικας περιέχει την τιμή στόχο και την πράξη της κλίκας, αν υπάρχει. Έπειτα εκτυπώνεται η λύση του προβλήματος, και σε κάθε κελί περιέχεται η τιμή που αντιστοιχεί στη λύση του puzzle.

Για να τρέξετε το πρόγραμμα εκτελέστε την εντολή **python3 kenken.py** και ακολουθήστε τις οδηγίες menu.

3. Οι αλγόριθμοι (BT, BT+MRV,FC, FC+MRV, MAC) εκτελέστηκαν μέσω της υλοποίησης που ζητήθηκε στο ερώτημα 2 σε μια εύκολη πίστα 3x3, σε μια δύσκολη 5x5, στην πίστα 6x6 που δώθηκε στην εκφώνηση της άσκησης και μια πολύ δύσκολη πίστα μεγέθους 9x9. Για να συγκριθεί η αποδοτικότητα των αλγορίθμων αυτών χρησιμοποιήθηκε ως κριτήριο αξιολόγησης ο Χρόνος εκτέλεσης του κάθε αλγορίθμου, ο αριθμος των αναθέσεων που κάναμε στις μεταβλητές του προβλήματος, και οι φορές που καλέστηκε η συνάρτηση περιορισμών, για τους παραπάνω χάρτες.

Για το 3x3:

Αλγόριθμοι	**Χρόνος Εκτέλεσης**	**Number of Assignments**	**Number of Conflict Calls**
BT	0.0006861	7	68
BT+MRV	0.0007839	5	71
FC	0.00069761	7	68
FC+MRV	0.0008041	5	67
MAC	0.001012	6	122

Για τον 5x5:

Αλγόριθμοι	**Χρόνος Εκτέλεσης**	**Number of Assignments**	**Number of Conflict Calls**
BT	0.029	60	1408
BT+MRV	0.02898	56	2286

FC	0.01039	20	599
FC+MRV	0.008	24	899
MAC	0.0336	15	2256

Για το 6x6:

Αλγόριθμος	**Χρόνος Εκτέλεσης**	**Number of Assignments**	**Number of Conflict Calls**
BT	0.04287	59	3332
BT+MRV	0.03724	39 (και αλλες τιμες +-30)	4330
FC	0.02894	39	1599
FC +MRV	0.02677	15	959
MAC	0.04285	15	3580

Για το 9x9:

Αλγόριθμος	**Χρόνος Εκτέλεσης**	**Number of Assignments**	**Number of Conflict Calls**
BT	34.0722	19402	4737811
BT+MRV	86.4473	49392	17291014
FC	3.0871	3583	498278
FC+MRV	0.1715	57	20653
MAC	17.186	208	3016679

Όπως παρατηρούμε ο αλγόριθμος υπαναχώρησης **BT** τρέχει πιο αργά από τους περισσότερους αλγορίθμους που εξετάσαμε πειραματικά (όπως φαίνεται στους παραπάνω πίνακες μετρήσεων). Αυτό συμβαίνει γιατί ο αλγόριθμος επιλέγει τυχαία την μεταβλητή στην οποία θα αναθέσει τιμή και η τιμή που θα της αναθέσει πέρα από την συνέπεια δεν έχει κάποιο άλλο κριτήριο καταλληλότητας. Γι αυτό το λόγο και παρατηρούμε πως γίνονται πολλές αναθέσεις τιμών αλλά και

κλήσεις της συνάρτησης περιορισμών, επειδή ακριβώς δεν υπάρχει κριτήριο ώστε να μπορέσουμε να αποφύγουμε ένα μεγάλο μέρος της υπαναχώρησης.

Επίσης με βάση τα πειραματικά αποτελέσματα παρατηρούμε πως ο αλγόριθμος **BT** με **MRV** είναι άλλες φορές πιο αργός άλλες φορές πιο γρήγορος απ' τον **BT** χωρίς MRV. Συγκεκριμένα παρατηρούμε η συνάρτηση περιορισμού καλείται περισσότερες φορές στην BT + MRV και οi μικρές βελτιώσεις σε χρόνους για μικρα puzzles ωφείλονται στις λιγότερες αναθέσεις τιμών. Αυτό συμβαίνει γιατί σαν στρατηγική ο BT + MRV δεν είναι έξυπνα σχεδιασμένος ώστε να δίνει καλύτερα αποτελέσματα.Η λογική του είναι απλώς να διαλέγει την μεταβλητή που έχει εκείνη τη στιγμή τις λιγότερες νόμιμες θέσεις το οποίο από μόνο του δεν έχει τη δυνατότητα να εξασφαλίσει μείωση της υπαναχώρησης κατά την εκτέλεση του BT και συχνά οδηγεί σε περιττές αναθέσεις τιμών και κλήσεις της συνάρτησης περιορισμών. Ειδικά στο 9x9 puzzle τόσο οι αναθέσεις, όσο και οι κλήσεις της συνάρτησης περιορισμών είναι σχεδόν τετραπλάσιες απ' οτι του απλού BT. Επίσης τα αποτελέσματα του αλγορίθμου αυτού είναι υπολογιστικά ασταθή δεδομένου οτι σε puzzles όπως το 6x6 δεν έχουμε σταθερό αριθμό αναθέσεων τιμών. Άρα καταλλήγουμε πως ο αλγόριθμος αυτός δεν επιφέρει προφανή βελτίωση έναντι του απλού BT.

Ο αλγόριθμος FC όπως παρατηρούμε πειραματικά προσφέρει εμφανή βελτίωση στην αποδοτικότητα του σε σχέση με τον BT. Τόσο ο χρόνος, όσο και οι αναθέσεις τιμών και οι έλεγχοι περιορισμών μειώνονται δραματικά. Κάτι τέτοιο είναι λογικό δεδομένου του μηχανισμού του FC που είναι το forward checking. Δηλαδή κάθε φορά που αναθέτει μια τιμή ελέγχει διαγράφοντας τις τιμές των υπόλοιπων μεταβλητών που είναι ασυνεπείς με την τιμή αυτής της μεταβλητής και αποτυγχάνει αν κάποια μεταβλητή μείνει με κενό domain. Αυτό βοηθάει αισθητά στην απόδοση αφού κρίνει αν μια ανάθεση είναι κατάλληλη σε σχέση με το σύνολο των υπολοίπων μεταβλητών του προβλήματος γλιτώνοντας έτσι πολύ χρόνο, αφού προειδοποιεί εγκαίρως για τις ακατάλληλες τιμες. Για παράδειγμα στο 9x9 puzzle τόσο ο χρόνος, οσο και οι αναθέσεις και οι κλήσεις της συνάρτησης περιορισμού σχεδόν υποδεκαπλασιάζονται. Επομένως ο αλγόριθμος FC είναι καλύτερος απ τον BT.

Ο αλγόριθμος FC+MRV όπως παρατηρούμε πειραματικά προσφέρει ακόμα καλύτερη βελτίωση και από τον αλγόριθμο FC. Αυτό συμβαίνει διότι προσθέτοντας τον μηχανισμό του MRV στον FC επιλέγουμε να αναθέσουμε τιμή στην μεταβλητή που έχει το μικρότερο πεδίο τιμών. Αυτό έχει ως αποτέλεσμα να υπάρχει μεγαλύτερη πιθανότητα να βρούμε μια τιμή που να ικανοποιεί τα constraints και το Forward checking, γρηγορότερα απ' οτι σε ένα απλό FC, αλλά και να εντοπίσουμε πιο γρήγορα αν μια λύση που ακολουθεί το δέντρο του προβλήματος δεν ειναι έγκυρη. Αυτό έχει ως αποτέλεσμα οι χρόνοι του να είναι αισθητά καλύτεροι σε σχέση με τον FC, αλλά και οι αναθέσεις τιμών και οι κλήσεις της συνάρτησεις περιορισμών. Ειδικά αν πάρουμε ως παράδειγμα το puzzle 9x9 βλέπουμε πως ο χρόνος λύσεις του προβλήματος, οι αναθέσεις και οι κλήσεις της συνάρτησης περιορισμών μειώνονται σε πολύ μεγάλο βαθμό, κάνοντας ένα δύσκολο πρόβλημα που πριν έπαιρνε αρκετό χρόνο να λύνεται σχεδόν στιγμιαία. Παρ' όλα αυτά υπάρχει μια μικρή πιθανότητα ο MRV να βλάψει την αποδοτικότητα του FC για ορισμένα προβλήματα, δεδομένου ότι πολλές φορές η επιλογή των στοιχείων να μην οδηγήσει γρηγορότερα σε backtracking αν ειναι λάθος αυτή η λύση.

Ο αλγόριθμος **mac** όπως παρατηρούμε από τα παραπάνω πειράματα είναι σχετικά αποδοτικός αλγόριθμος σε σχέση με τον BT αλλά όχι γρηγορότερος από αλγορίθμους όπως ο FC ή ο FC + MRV. Αυτό συμβαίνει γιατί ο mac κάθε φορά που αναθέτει μια τιμή σε μια μεταβλητή ελέγχει αν διατηρείται η συνέπεια ακμης μεταξύ της μεταβλητής και των γειτόνων της μέσω του AC-3. Αν εντοπιστεί ασυνέπεια τότε προχωρούμε στην αναθεση αλλης τιμής στη μεταβλητή. Αυτό έχει ως αποτέλεσμα να κάνουμε λίγες αναθέσεις τιμών γιατί ελέγχονται για ασυνέπεια σε σχέση με τους

γείτονες των μεταβλητών τους, αλλά πολλές κλήσεις της συνάρτησης περιορισμών κάτι που μειώνει την αποδοτικότητα του αλγορίθμου. Άρα για προβλήματα όπως το 9χ9 βλέπουμε πως ο χρόνος εκτέλεσης μειώνεται μόνο στο μισό απ' τον BT παρ'ολο που έχουμε λίγες αναθέσεις γιατί υπάρχουν πολλοί ελεγχοι περιορισμών.

4)

MAP	**TIME**	**NASSIGNMENTS**	**NCOSTRAINTCALLS**	**SOLUTION**
Easy 3x3	0.00109	6	106	**yes**
Hard 5x5	0.059/118.96	43/100012	8176/24197740	**yes/no**
Exercise Map 6x6	0.07393.53	21/100015	4685/63330533	**yes/no**
Expert 9x9	**n/a**	**n/a**	**n/a**	**no**

Στον παραπάνω πίνακα φαίνονται οι πειραματικές εκτελέσεις του αλγορίθμου min conflicts. Παρατηρούμε ότι ο αλγόριθμος min_conflicts δίνει αποτελέσματα σε μικρά προβλήματα, ενώ σε μεσσαίου μεγέθους άλλες φορές δίνει λυσεις σε ικανοποιητικό χρόνο και άλλες φορές δε δίνει λύση. Σε μεγάλου μεγέθους όπως το expert 9x9 δεν δίνει λύση. Τις φορές που δεν έχουμε λύση στα hard και exercise map αυτό που συμβαίνει είναι ότι ουσιαστικά ο αλγόριθμος ξεπερνά τον μέγιστο αριθμό βημάτων που του έχουν οριστεί.

Ο ευρετικός αλγόριθμος min conflicts αναθέτει τυχαία τιμές στις μεταβλητές του προβλήματος και αν αυτή η ανάθεση ειναι λύση την επιστρέφει, αλλιώς προσπαθεί για κάθε μεταβλητή του προβλήματος να επιλέξει την τιμή που θα ελαχιστοποιήσει τον αριθμό των συγκρούσεων με άλλες μεταβλητές. Φυσικά όπως αναφέρθηκε παραπάνω αν ξεπεραστεί ο αριθμός των βημάτων ο αλγόριθμος θα σταματήσει χωρίς να επιστρέψει κάποια λύση.

Επειδή κανει μια τυχαία ανάθεση τιμών στην αρχή όπως αναφέρθηκε προηγουμένως, είναι λογικό μερικές φορές να δίνεται λύση και άλλες όχι. Και αυτό γιατί υπαχει περίπτωση ο αλγόριθμος να συνεχίσει να ψάχνει για λύση χωρίς να βρίσκει και να ξεπεραστεί ο μέγιστος αριθμός βημάτων και αυτό γιατί δεν λαμβάνει υπόψιν τις μερικές λύσεις που είχε υπολογίσει προηγουμένως, που θα τον βοηθούσαν στο να καταλήξει σε λύση, όπως οι άλλοι αλγόριθμοι που εξετάσαμε.

Άρα δεν μπορεί να θεωρηθεί πλήρης όπως όλοι όσοι εξετάσαμε προηγουμένως. Τέλος μπορούμε να πούμε πως οι λύσεις του προβλήματος είναι αραιά κατανεμημένες στο χώρο των καταστάσεων άρα ο αλγόριθμος δυσκολεύεται αρκετά μέσω της προσέγγισης του να βρει πλήρη λύση στο πρόβλημα του Kenken Puzzle.

Συνεπώς ο αλγόριθμος min_conflicts ενώ ισχυρός για άλλα προβλήματα CSP στο συγκεκριμένο δεν ειναι ιδιαίτερα αποδοτικός ώστε να προτιμηθεί έναντι των άλλων που εξετάσαμε.