ΙΟΥΝΙΟΣ 2023

# ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ

ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ

ΓΡΑΜΜΙΚΗ ΚΑΙ ΣΥΝΔΥΑΣΤΙΚΗ

ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ

ΥΠΕΥΘΥΝΗ ΚΑΘΗΓΗΤΡΙΑ: ΔΑΣΚΑΛΑΚΗ ΣΟΦΙΑ

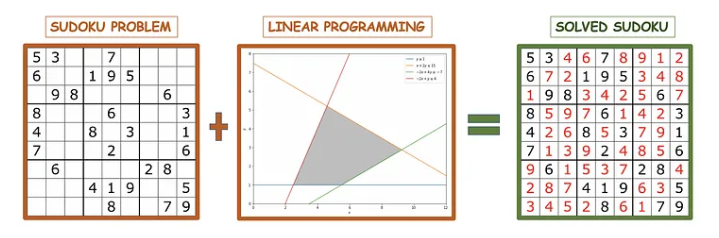
ΓΙΑΝΝΑΚΑΚΗΣ ΑΝΑΣΤΑΣΙΟΣ

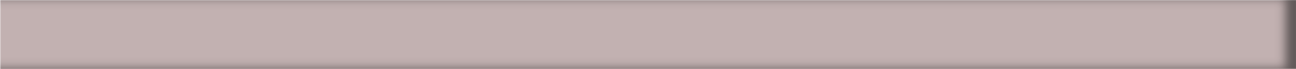
1072905

4ο ΈΤΟΣ

***ΘΕΜΑ***

***Λύση Sudoku παζλ με χρήση γραμμικού προγραμματισμού***





ΠΕΡΙΓΡΑΦΗ ΚΑΙ ΣΚΟΠΟΣ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ

1. **Περιγραφή του προβλήματος**

Το παζλ Sudoku είναι ένα δημοφιλές παιχνίδι παζλ που βασίζεται στη λογική και περιλαμβάνει τη συμπλήρωση ενός πλέγματος 9x9 με ψηφία από το 1 έως το 9, έτσι ώστε κάθε γραμμή, στήλη και υποπλέγμα 3x3 να περιέχει κάθε ψηφίο ακριβώς μία φορά. Το παιχνίδι παίζεται συχνά για διασκέδαση και ψυχαγωγία, αλλά έχει επίσης προσελκύσει την προσοχή των ερευνητών της επιστήμης των υπολογιστών και της επιχειρησιακής έρευνας, οι οποίοι έχουν διερευνήσει τις μαθηματικές του ιδιότητες και έχουν αναπτύξει αλγορίθμους για την επίλυση παζλ Sudoku.

Από υπολογιστική άποψη, οι γρίφοι Sudoku μπορούν να θεωρηθούν ως προβλήματα ικανοποίησης περιορισμών, όπου ο στόχος είναι να βρεθεί μια λύση που να ικανοποιεί ένα σύνολο περιορισμών. Οι περιορισμοί σε ένα παζλ Sudoku είναι απλοί: κάθε γραμμή, στήλη και υποπλέγμα πρέπει να περιέχει κάθε ψηφίο ακριβώς μία φορά. Ωστόσο, το παζλ γίνεται πρόκληση επειδή οι περιορισμοί είναι αλληλένδετοι και επιβάλλουν υψηλό βαθμό αμοιβαίας αποκλειστικότητας μεταξύ των ψηφίων στο πλέγμα.

Οι γρίφοι Sudoku κυκλοφορούν σε διάφορα επίπεδα δυσκολίας, με ορισμένους γρίφους να απαιτούν σύνθετη συλλογιστική και δεξιότητες εξαγωγής συμπερασμάτων για την επίλυση, ενώ άλλοι μπορούν να λυθούν με τη χρήση βασικής λογικής και δοκιμής και λάθους. Η δυσκολία ενός γρίφου εξαρτάται από διάφορους παράγοντες, όπως ο αριθμός των ενδείξεων που δίνονται στην αρχή, η κατανομή των ενδείξεων στο πλέγμα και η τοποθέτηση των ενδείξεων σε σχέση μεταξύ τους.

Οι γρίφοι Sudoku έχουν αρκετές εφαρμογές πέρα από την ψυχαγωγία. Για παράδειγμα, έχουν χρησιμοποιηθεί ως σημεία αναφοράς για τη δοκιμή αλγορίθμων στην τεχνητή νοημοσύνη, τη βελτιστοποίηση και τον προγραμματισμό με περιορισμούς. Έχουν επίσης χρησιμοποιηθεί ως εκπαιδευτικά εργαλεία για τη διδασκαλία της λογικής, της επίλυσης προβλημάτων και των μαθηματικών σε παιδιά και ενήλικες.

Συνολικά, το παζλ Sudoku είναι ένα ενδιαφέρον και δύσκολο πρόβλημα που μπορεί να προσεγγιστεί από διάφορες οπτικές γωνίες, από τη λογική και την εξαγωγή συμπερασμάτων έως τη βελτιστοποίηση και τον προγραμματισμό περιορισμών. Προσφέρει ένα πλούσιο και ποικίλο σύνολο ερευνητικών και εφαρμοστικών ευκαιριών, καθιστώντας το ένα δημοφιλές θέμα στους τομείς της επιστήμης των υπολογιστών, της επιχειρησιακής έρευνας και των μαθηματικών.

**Στο συγκεκριμένο πρότζεκτ θα εξετάσουμε τις λύσεις διαφόρων ειδών Sudoku, ορίζοντας κάθε φορά διαφορετικούς περιορισμούς.**

1. **Σκοπιμότητα**

Το Sudoku είναι ένα εφικτό πρόβλημα, που σημαίνει ότι μπορεί να επιλυθεί αποτελεσματικά και αποδοτικά. Αυτό συμβαίνει επειδή το Sudoku είναι ένα καλά καθορισμένο πρόβλημα με πεπερασμένο αριθμό πιθανών λύσεων. Το πρόβλημα Sudoku μπορεί να αναπαρασταθεί ως πρόβλημα ικανοποίησης περιορισμών, το οποίο μπορεί να επιλυθεί με τη χρήση διαφόρων αλγορίθμων, όπως η ωμή βία, η οπισθοδρόμηση και η διάδοση περιορισμών.

Μια προσέγγιση για την επίλυση του Sudoku είναι η χρήση ενός αλγορίθμου οπισθοδρόμησης, όπου ο αλγόριθμος προσπαθεί να γεμίσει κάθε κελί του πλέγματος Sudoku με έναν έγκυρο αριθμό, ξεκινώντας από το πάνω αριστερό κελί και κινούμενος προς τα δεξιά και κάτω. Εάν ένα κελί δεν μπορεί να συμπληρωθεί με έγκυρο αριθμό, ο αλγόριθμος επιστρέφει στο προηγούμενο κελί και δοκιμάζει έναν διαφορετικό αριθμό. Αυτή η διαδικασία συνεχίζεται μέχρι να συμπληρωθεί πλήρως το πλέγμα ή να διαπιστωθεί ότι δεν υπάρχει έγκυρη λύση.

Μια άλλη προσέγγιση είναι η χρήση ενός αλγορίθμου διάδοσης περιορισμών, όπως ο αλγόριθμος συνέπειας τόξου, ο οποίος εξαλείφει τις ασυνεπείς τιμές από την περιοχή κάθε κελιού με βάση τους περιορισμούς που επιβάλλονται από τα γειτονικά κελιά. Η διαδικασία αυτή συνεχίζεται έως ότου είτε βρεθεί έγκυρη λύση είτε διαπιστωθεί ότι δεν υπάρχει έγκυρη λύση.

Ο γραμμικός προγραμματισμός μπορεί επίσης να χρησιμοποιηθεί για την επίλυση προβλημάτων Sudoku με τη διατύπωση του προβλήματος ως προβλήματος ακέραιου προγραμματισμού, όπου η αντικειμενική συνάρτηση ελαχιστοποιείται με βάση ένα σύνολο περιορισμών. Οι μεταβλητές απόφασης σε αυτή την περίπτωση αντιπροσωπεύουν τις τιμές σε κάθε κελί του πλέγματος Sudoku. Στο πρότζεκτ αυτό εμείς, θα ασχοληθούμε με την συγκεκριμένη προσέγγιση. Στον κώδικα χρησιμοποιείται η βιβλιοθήκη της Python εν ονόματι Pulp, η οποία εισάγει τον γραμμικό προγραμματισμό ως λύση σε ένα πρόβλημα.

Συμπερασματικά, το Sudoku είναι ένα εφικτό πρόβλημα που μπορεί να επιλυθεί αποτελεσματικά με τη χρήση μιας ποικιλίας αλγορίθμων, συμπεριλαμβανομένων της οπισθοδρόμησης, της διάδοσης περιορισμών και του ακέραιου προγραμματισμού.

ΜΟΝΤΕΛΟΠΟΙΗΣΗ

1. **Μοντελοποίηση**

Αρχικά:

***Τι είναι ο γραμμικός προγραμματισμός;***

Ο γραμμικός προγραμματισμός είναι ένα μοντέλο βελτιστοποίησης με περιορισμούς που έχει 3 βασικά συστατικά: αντικειμενική συνάρτηση, μεταβλητές απόφασης και περιορισμούς.

***Αντικειμενική συνάρτηση:***

Η αντικειμενική συνάρτηση στη συγκεκριμένη περίπτωση για κάθε είδος Sudoku, παραμένει ίδια και χωρίς ουσία, καθώς το πρόβλημα μας δεν χρειάζεται μεγιστοποίηση ή ελαχιστοποίηση, αλλά απλά επίλυση.

***Μεταβλητές απόφασης:***

Όπως θα φανεί και στον κώδικα παρακάτω, για κάθε είδος έχουμε μια μεταβλητή grid\_vars(row,column,value), στην οποία αναθέτουμε τη σειρά, τη στήλη και τέλος την τιμή που έχει (1-9).

***Βασικοί Περιορισμοί για κάθε είδος:***

Κάθε κελί μπορεί να περιέχει μόνο έναν αριθμό.

Κάθε αριθμός μπορεί να εμφανίζεται μόνο μία φορά σε κάθε γραμμή.

Κάθε αριθμός μπορεί να εμφανιστεί μόνο μία φορά σε κάθε στήλη.

Κάθε αριθμός μπορεί να εμφανιστεί μόνο μία φορά σε κάθε πλαίσιο.

Οι αρχικές τιμές πρέπει να τηρούνται για όλα τα κελιά όπου η τιμή τους δεν είναι 0.

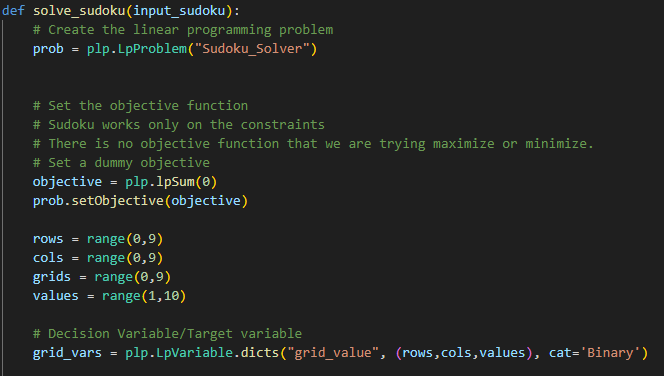
Αυτοί είναι οι βασικοί περιορισμοί που χρησιμοποιούμε για κάθε είδος Sudoku που θα προσπαθήσουμε να επιλύσουμε παρακάτω.

ΕΙΔΗ SUDOKU – ΠΕΡΙΓΡΑΦΗ/ΕΠΙΛΥΣΗ - ΚΩΔΙΚΑΣ

* 1. **Normal Sudoku**

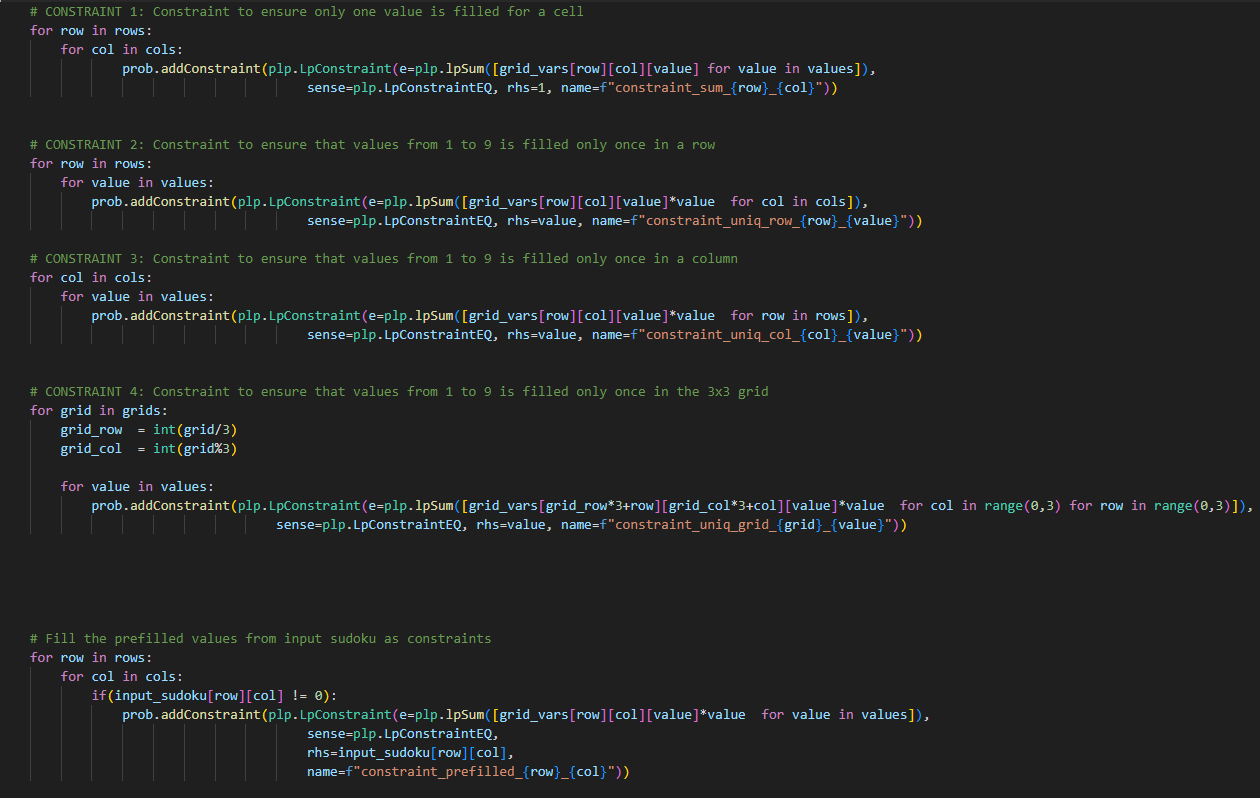
Για τη μοντελοποίηση του απλού κανονικού Sudoku θα αρκεστούμε στους απλούς περιορισμούς και στις μεταβλητές που αναφέραμε πριν.

Από τον παρακάτω κώδικα:



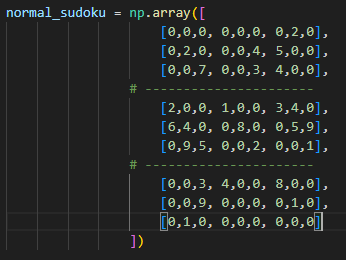
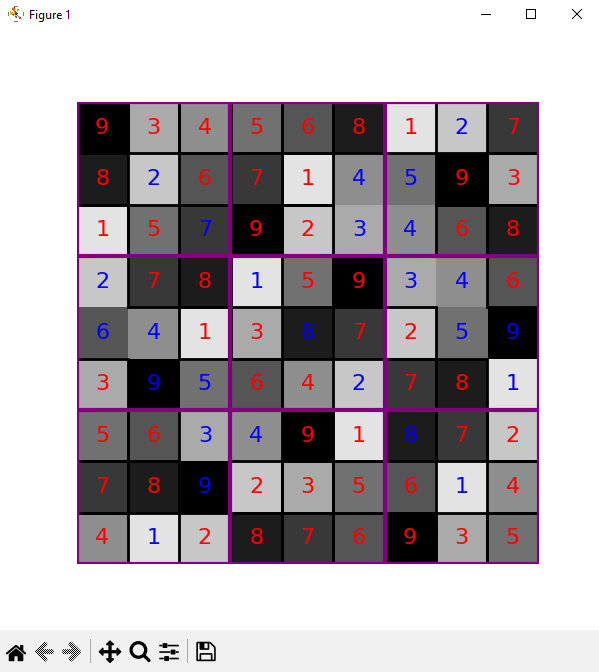
Ορίζονται οι μεταβλητές και το πρόβλημα. Όπως αναφέραμε η αντικειμενική μας συνάρτηση δεν έχει ουσία αφού δεν ψάχνουμε για minimization/maximization.

Η μεταβλητή grid\_vars δέχεται ως ορίσματα rows,cols,values. Στην ουσία αναφέρεται σε κάθε ξεχωριστό κελί στο Sudoku μας.

Έπειτα εισάγουμε τους περιορισμούς:

* Κάθε κελί μπορεί να περιέχει μόνο έναν αριθμό.
* Κάθε αριθμός μπορεί να εμφανίζεται μόνο μία φορά σε κάθε γραμμή.
* Κάθε αριθμός μπορεί να εμφανιστεί μόνο μία φορά σε κάθε στήλη.
* Κάθε αριθμός μπορεί να εμφανιστεί μόνο μία φορά σε κάθε πλαίσιο.
* Οι αρχικές τιμές πρέπει να τηρούνται για όλα τα κελιά όπου η τιμή τους δεν είναι 0.

Και με τη χρήση της βιβλιοθήκης matplotlib, το πρόγραμμα παίρνει ως είσοδο ένα αρχικό grid Sudoku, άλυτο προφανώς, και εμφανίζει το αποτέλεσμα όπως φαίνεται παρακάτω:



Όπου οι μπλε αριθμοί είναι αυτοί που προϋπήρχαν

και οι κόκκινοι αυτή που μπήκαν στα κελιά μετά τη

χρήση του γραμμικού προγραμματισμού. Όπως

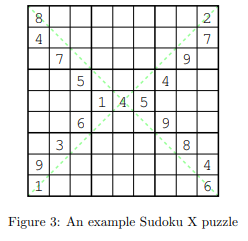
βλέπουμε στο αποτέλεσμα στα 3x3 κουτιά που

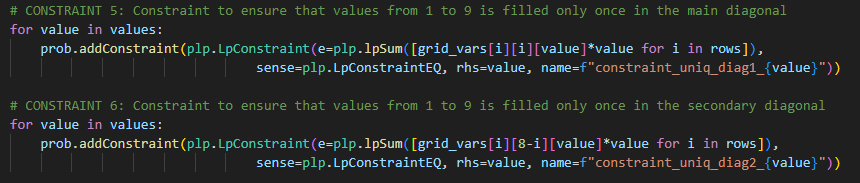
ορίζουν οι μωβ γραμμές υπάρχει μοναδικότητα από

1-9, καθώς και σε κάθε γραμμή και στήλη ολόκληρου του Sudoku.

* 1. **Sudoku X**

Το παζλ "Sudoku X" είναι σαν το κανονικό παζλ, με μια επιπλέον απαίτηση: οι δύο μεγάλες διαγώνιες του πίνακα πρέπει επίσης να περιέχουν κάθε ψηφίο από το 1 έως το 9 ακριβώς μία φορά. Έτσι, κάθε λύση στο Sudoku X παζλ είναι επίσης μια λύση στο τυπικό παζλ Sudoku, αλλά το αντίστροφο δεν ισχύει. Ένα παράδειγμα ενός Sudoku X δίνεται στο Σχήμα 3, όπου κάθε μεγάλη διαγώνιος περιέχει μια διακεκομμένη πράσινη γραμμή.



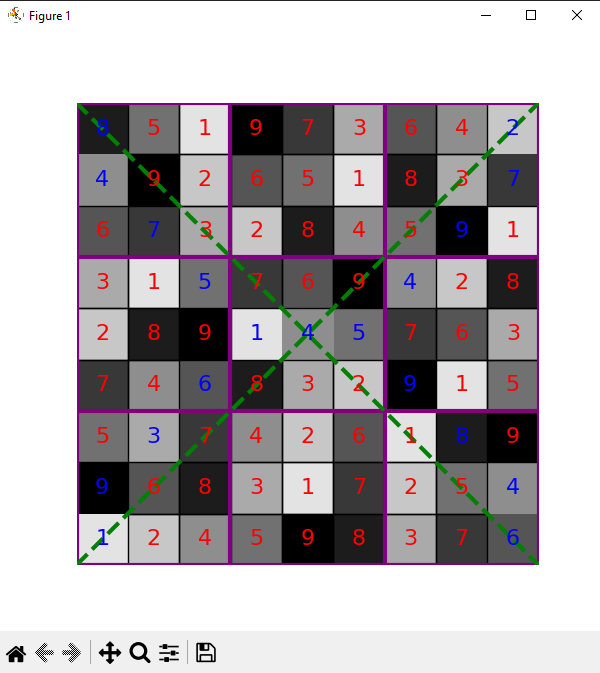
Για την υλοποίηση αυτών των δύο έξτρα περιορισμών χρησιμοποιούμε τον παρακάτω κώδικα:

Ο κώδικας επαναλαμβάνει κάθε πιθανή τιμή στη λίστα τιμών (η οποία σε αυτή την περίπτωση είναι [1, 2, 3, 4, 5, 6, 7, 8, 9]. Για κάθε τιμή, δημιουργεί έναν νέο γραμμικό περιορισμό.

Αυτός ο περιορισμός επιβάλλει ότι τα διαγώνια κελιά στο πλέγμα Sudoku (δηλαδή τα κελιά όπου ο δείκτης γραμμής και ο δείκτης στήλης είναι ίδιοι) μπορούν να περιέχουν μόνο την καθορισμένη τιμή και ότι κάθε τιμή μπορεί να εμφανίζεται μόνο μία φορά κατά μήκος της διαγώνιου.

Η συνάρτηση **plp.LpConstraint** χρησιμοποιείται για τη δημιουργία ενός νέου αντικειμένου γραμμικού περιορισμού, το οποίο λαμβάνει διάφορα ορίσματα:

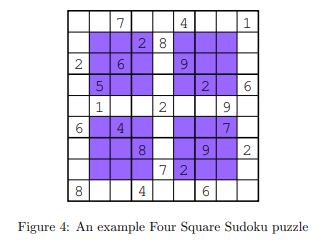
* **e**: η έκφραση που ορίζει τον γραμμικό περιορισμό (στην προκειμένη περίπτωση, ένα άθροισμα όρων που αντιπροσωπεύουν τα διαγώνια κελιά του πλέγματος πολλαπλασιασμένα με την τιμή που περιορίζεται)
* **sense**: η κατεύθυνση του περιορισμού (στην προκειμένη περίπτωση, ισότητα, που υποδεικνύεται από το plp.LpConstraintEQ)
* **rhs**: η τιμή της δεξιάς πλευράς του περιορισμού (στην προκειμένη περίπτωση, η καθορισμένη τιμή)
* **name**: ένα μοναδικό όνομα για τον περιορισμό (στην προκειμένη περίπτωση, μια συμβολοσειρά που περιγράφει τον περιορισμό)

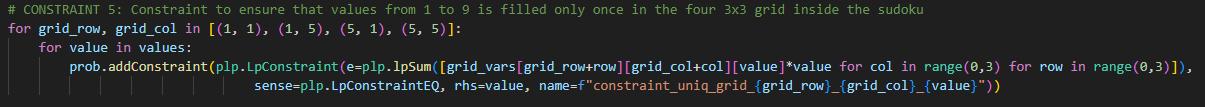
Παρακάτω φαίνεται το αποτέλεσμα του κώδικα μας και η λύση του Sudoku της φωτογραφίας:

**Φαίνονται επίσης και οι διαγώνιοι στις οποίες υπάρχει μοναδικότητα των αριθμών 1-9!**

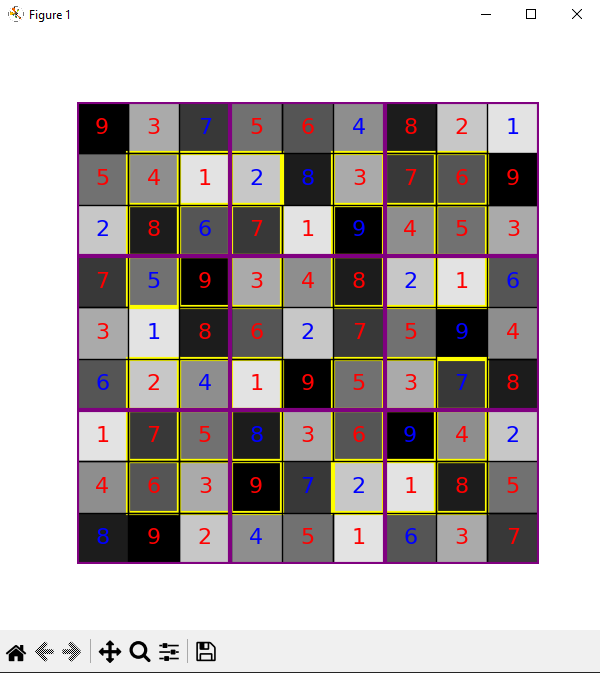
* 1. **Four Square Sudoku**

Το Sudoku "Τέσσερα τετράγωνα" είναι και πάλι ένα τυπικό παζλ, αλλά με μια επιπλέον απαίτηση. Υπάρχουν 4 σκιασμένες περιοχές 3 × 3 στον πίνακα Sudoku, και επιπλέον των απαιτήσεων του τυπικού Sudoku, κάθε σκιασμένη περιοχή πρέπει επίσης να περιέχει κάθε ψηφίο από το 1 έως το 9 ακριβώς μία φορά. Ένα δείγμα παζλ τεσσάρων τετραγώνων είναι το εξής δίνεται στην εικόνα 4.



Προσθέτοντας στον κώδικα μας τον παρακάτω περιορισμό:

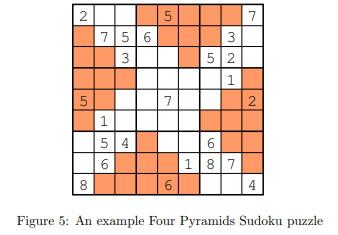
Αυτές οι γραμμές κώδικα ορίζουν έναν περιορισμό που εξασφαλίζει ότι κάθε τιμή από το 1 έως το 9 εμφανίζεται μόνο μία φορά σε κάθε ένα από τα τέσσερα υποπλέγματα 3x3 μέσα στο πλέγμα Sudoku. Ο κώδικας επαναλαμβάνει τα τέσσερα υπο-πλέγματα που βρίσκονται στις θέσεις (1,1), (1,5), (5,1) και (5,5) και στη συνέχεια, για κάθε υπο-πλέγμα, επαναλαμβάνει κάθε τιμή από το 1 έως το 9. Για κάθε τιμή, ο κώδικας δημιουργεί έναν περιορισμό χρησιμοποιώντας τη συνάρτηση LpConstraint της βιβλιοθήκης PuLP. Η αριστερή πλευρά (LHS) του περιορισμού ορίζεται χρησιμοποιώντας τη συνάρτηση lpSum, η οποία υπολογίζει το άθροισμα του γινομένου των μεταβλητών απόφασης και των τιμών τους για κάθε κελί του υποπλέγματος. Το λεξικό grid\_vars αποθηκεύει τις μεταβλητές απόφασης για κάθε κελί, όπου κάθε μεταβλητή αντιπροσωπεύει το αν μια συγκεκριμένη τιμή αποδίδεται στο κελί. Το LHS του περιορισμού, επομένως, εξασφαλίζει ότι κάθε τιμή εμφανίζεται μόνο μία φορά στο υπο-πλέγμα.

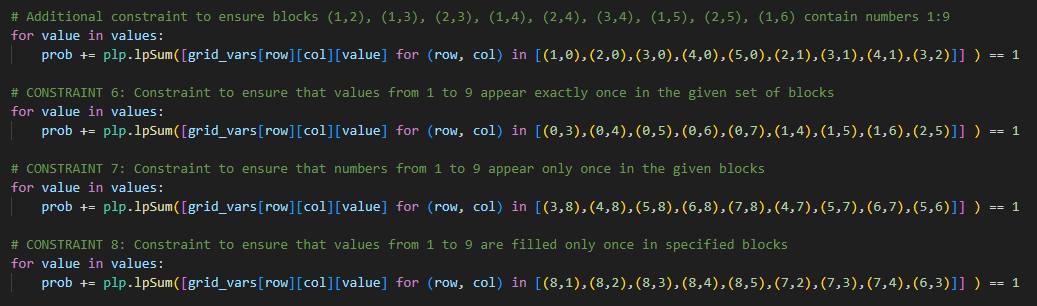
Συνεπώς η λύση του παραπάνω Sudoku είναι:

**Και τα κίτρινα κουτάκια υποδεικνύουν τα υποπλέγματα που θα πρέπει να υπακούν στον επιπλέον περιορισμό!**

* 1. **Four Pyramids Sudoku**

Όπως και η παραλλαγή των τεσσάρων τετραγώνων, το Sudoku "Τέσσερις πυραμίδες" είναι ένα τυπικό παζλ Sudoku με τέσσερις σκιασμένες περιοχές, όπου πρέπει να διασφαλιστεί ότι κάθε σκιασμένη περιοχή περιέχει ακριβώς ένα από κάθε ψηφίο από το 1 έως το 9 (εκτός από την ικανοποίηση των απαιτήσεων του τυπικού Sudoku). Όπως μπορεί κανείς να δει στο παράδειγμα παρακάτω, οι σκιασμένες περιοχές έχουν σχήμα κάπως σαν πυραμίδες, δίνοντας σε αυτή την παραλλαγή το όνομά της.

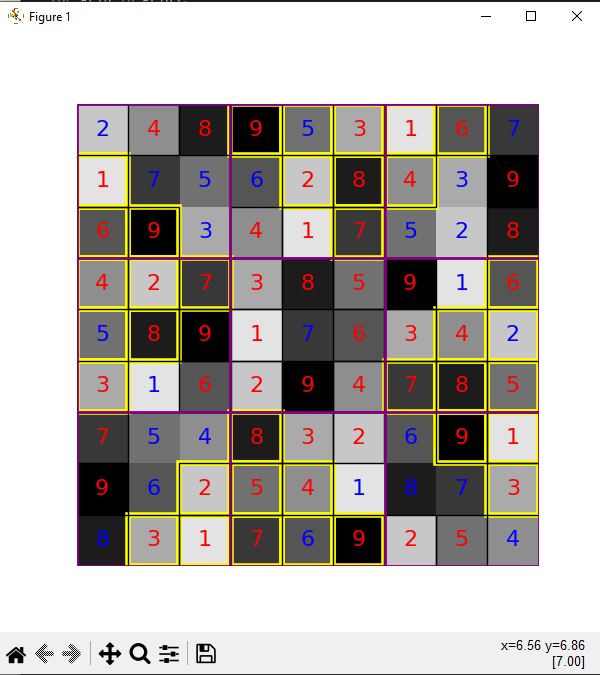


Προσθέτοντας στον κώδικα μας τον παρακάτω περιορισμό:

Ο κώδικας αρχικά επαναλαμβάνει τους αριθμούς 1 έως 9 χρησιμοποιώντας τη μεταβλητή 'value'. Για κάθε τιμή, ο περιορισμός δημιουργείται με τη χρήση του τελεστή +=, ο οποίος προσθέτει τον περιορισμό στο υπάρχον πρόβλημα βελτιστοποίησης 'prob'.

Ο περιορισμός εκφράζεται χρησιμοποιώντας τη συνάρτηση plp.lpSum(), η οποία αθροίζει τις τιμές της 'grid\_vars' για τα καθορισμένα ζεύγη (row, col) χρησιμοποιώντας μια κατανόηση λίστας. Αυτά τα ζεύγη (row, col) αντιπροσωπεύουν τα κελιά που ανήκουν στα καθορισμένα μπλοκ 3x3. Ο περιορισμός απαιτεί το άθροισμα αυτών των τιμών να είναι ίσο με 1, υποδεικνύοντας ότι κάθε αριθμός από το 1 έως το 9 πρέπει να εμφανίζεται ακριβώς μία φορά στα καθορισμένα μπλοκ 3x3.

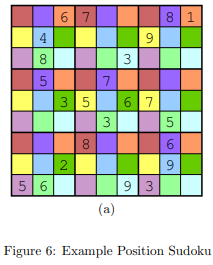
Συνοπτικά, αυτές οι γραμμές κώδικα δημιουργούν έναν περιορισμό που εξασφαλίζει ότι ορισμένα μπλοκ στο πλέγμα Sudoku περιέχουν κάθε αριθμό από το 1 έως το 9 ακριβώς μία φορά. Και κάθε φορά μπορούμε εμείς να ορίσουμε ποια μπλοκ θα είναι αυτά.

Συνεπώς η λύση του παραπάνω Sudoku είναι:

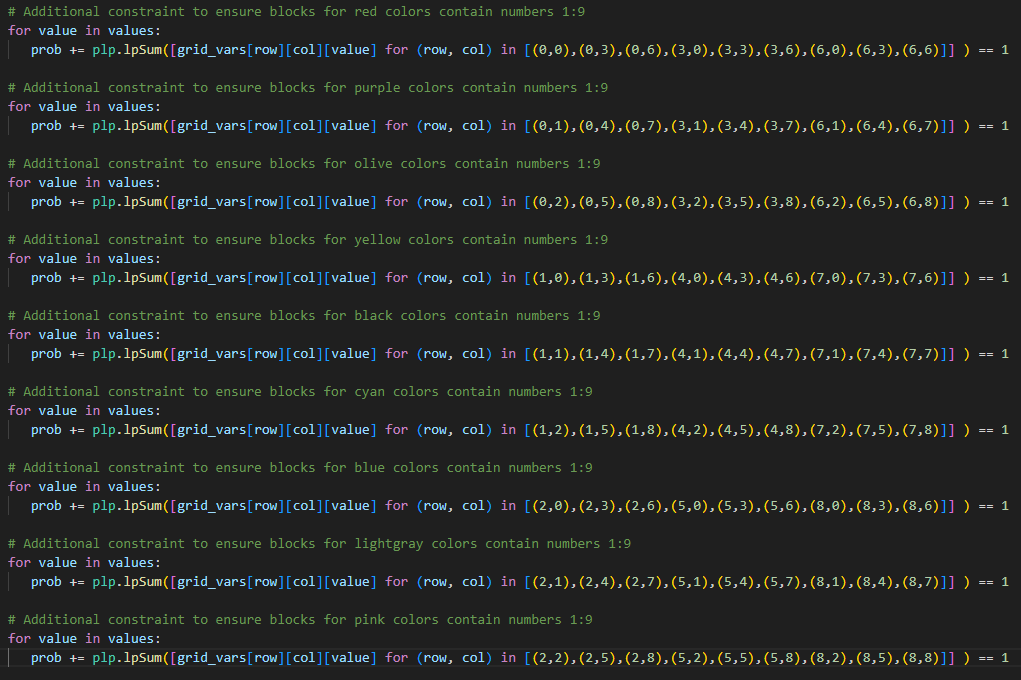
**Και τα κίτρινα κουτάκια υποδεικνύουν τις 4 πυραμίδες που θα πρέπει να υπακούν στον επιπλέον περιορισμό!**

* 1. **Position Sudoku**

Για κάθε στοιχείο σε ένα ταμπλό "Position Sudoku", όχι μόνο πρέπει να ληφθούν υπόψη το υποπλέγμα 3 × 3 μέσα στο οποίο βρίσκεται το στοιχείο, αλλά και τη θέση του στοιχείου μέσα στο 3 × 3 υποπλέγματος. Εκτός από την ικανοποίηση των τυπικών απαιτήσεων του Sudoku, μια λύση στο Position Sudoku πρέπει να είναι τέτοια ώστε ακριβώς ένα από κάθε ψηφίο από το 1 έως το 9 να περιέχεται στα πάνω αριστερά τετράγωνα και των εννέα 3 × 3 υποπλεγμάτων, ακριβώς ένα από κάθε ψηφίο να περιέχεται στα πάνω μεσαία τετράγωνα όλων των υποπλεγμάτων, και ούτω καθεξής. Στο δείγμα παζλ που βρίσκεται στο Σχήμα 6 (α), το σύνολο των τετραγώνων του πίνακα που μοιράζονται ένα δεδομένο χρώμα πρέπει να περιέχει κάθε ψηφίο ακριβώς μία φορά.

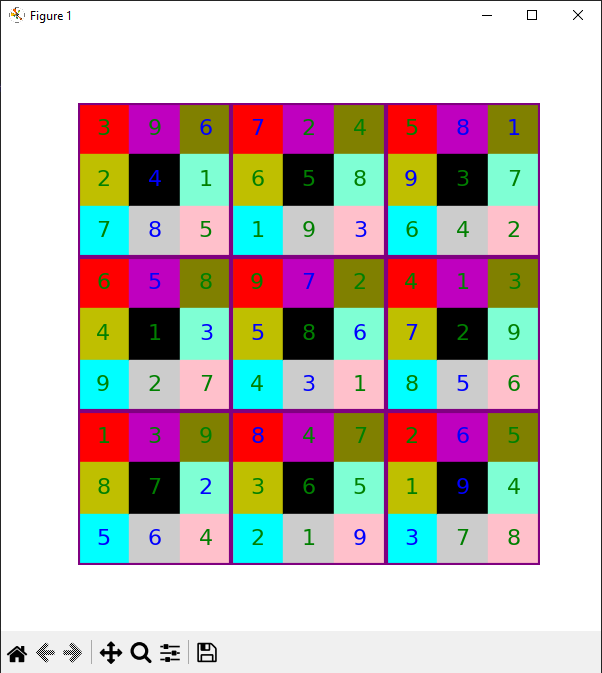


Προσθέτοντας στον κώδικα μας τους παρακάτω περιορισμούς:

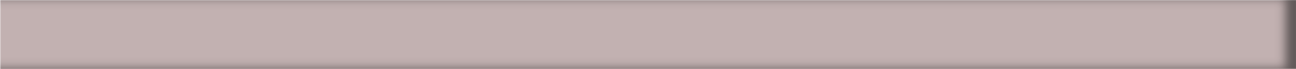


Αυτές οι γραμμές κώδικα προσθέτουν έναν πρόσθετο περιορισμό για να εξασφαλίσουν ότι τα μπλοκ που επισημαίνονται με κόκκινο χρώμα στο πλέγμα Sudoku περιέχουν αριθμούς από το 1 έως το 9 ακριβώς μία φορά. Ο περιορισμός προστίθεται χρησιμοποιώντας έναν βρόχο for που επαναλαμβάνει κάθε τιμή από το 1 έως το 9. Για κάθε τιμή, το άθροισμα των μεταβλητών στα αντίστοιχα κελιά των μπλοκ με το ίδιο χρώμα περιορίζεται να είναι ίσο με 1. Αυτό επιτυγχάνεται με τη χρήση της μεθόδου lpSum της μονάδας plp για να αθροίσει τις μεταβλητές grid\_vars που αντιστοιχούν στα κελιά των μπλοκ με το ίδιο χρώμα που περιέχουν την τρέχουσα τιμή.

Συνεπώς η λύση του παραπάνω Sudoku είναι:



**Και το κάθε χρώμα υποδεικνύει ένα υπό-πλέγμα του αρχικού μας Sudoku, στο οποίο θα πρέπει να υπάρχει μοναδικότητα των αριθμών 1-9. Όπως και γίνεται!**



ΤΡΟΠΟΙ ΕΠΙΛΥΣΗΣ SUDOKU

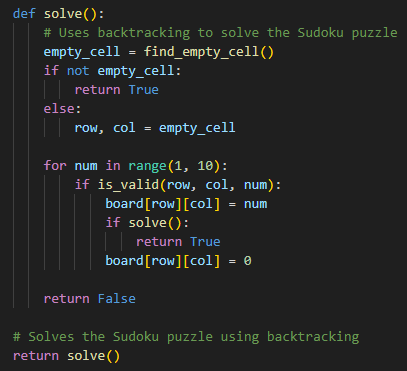
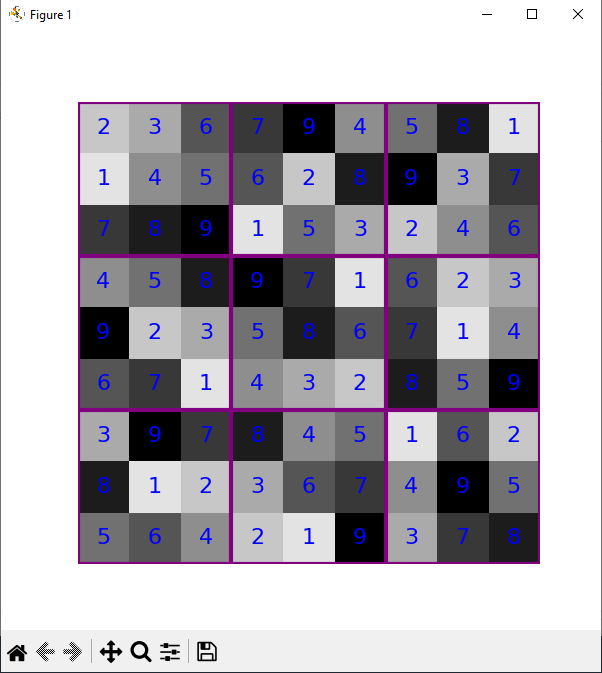
* 1. **Επίλυση με τη χρήση Backtracking**

Ο αλγόριθμος backtracking είναι μια κοινή τεχνική για την επίλυση προβλημάτων ικανοποίησης περιορισμών (CSPs) όπως το Sudoku. Η ιδέα πίσω από την οπισθοδρόμηση είναι να δοκιμάζονται μία προς μία όλες οι πιθανές τιμές για ένα κενό κελί, και αν μια τιμή παραβιάζει κάποιον από τους περιορισμούς, να γίνεται επιστροφή στο προηγούμενο κελί και να δοκιμάζεται μια διαφορετική τιμή. Αυτή η διαδικασία συνεχίζεται μέχρι να γεμίσουν όλα τα κελιά με έγκυρες τιμές ή να εξαντληθεί ο χώρος αναζήτησης. Ακολουθεί μια υψηλού επιπέδου επισκόπηση του τρόπου λειτουργίας του αλγορίθμου backtracking για το Sudoku:

* Βρείτε ένα κενό κελί στο πλέγμα Sudoku. Δοκιμάστε όλες τις πιθανές τιμές (1-9) για αυτό το κελί, μία κάθε φορά.
* Εάν μια τιμή είναι έγκυρη, μετακινηθείτε στο επόμενο κενό κελί και επαναλάβετε τα βήματα 1-2.
* Εάν καμία τιμή δεν είναι έγκυρη, επιστρέψτε στο προηγούμενο κελί και δοκιμάστε μια διαφορετική τιμή.
* Επαναλάβετε τα βήματα 1-4 μέχρι να λυθεί το Sudoku.

Για να επιταχυνθεί η αναζήτηση, η οπισθοδρόμηση χρησιμοποιεί συχνά μια τεχνική που ονομάζεται "διάδοση περιορισμών", η οποία περιορίζει τις πιθανές τιμές για κάθε κελί με βάση τις τιμές στα γειτονικά του κελιά. Αυτό μειώνει τον αριθμό των δυνατοτήτων προς διερεύνηση και μπορεί να βοηθήσει στην αποφυγή της αναζήτησης ολόκληρου του χώρου των πιθανών λύσεων. Συνολικά, η οπισθοδρόμηση είναι μια ισχυρή τεχνική για την επίλυση CSPs όπως το Sudoku, και μπορεί να εφαρμοστεί και σε ένα ευρύ φάσμα άλλων προβλημάτων.

Το **Backtracking** στον κώδικα γίνεται στην συνάρτηση παρακάτω, με δίπλα το αποτέλεσμα:

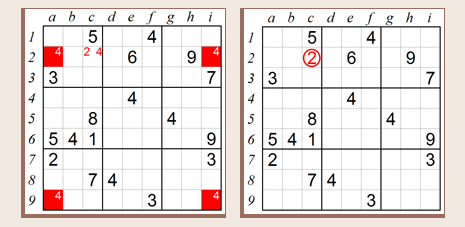




* 1. **Επίλυση με τη χρήση X-Wing Method**

Εξάλειψη τετραγώνων με τη χρήση του X-Wing:

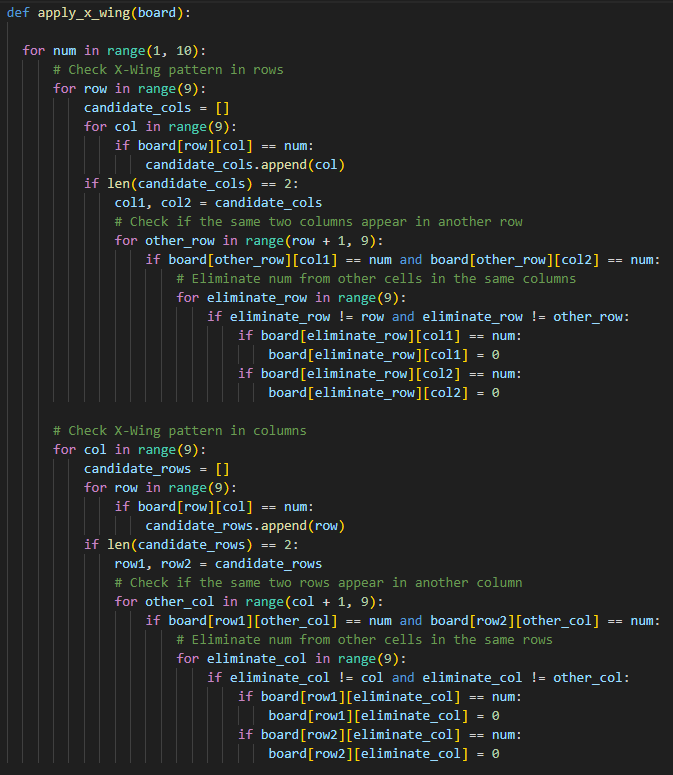
Η τεχνική X-Wing χρησιμοποιείται σε σπάνιες καταστάσεις που εμφανίζονται σε ορισμένους εξαιρετικά δύσκολους γρίφους. Σαρώνοντας τη στήλη a βλέπουμε ότι το 4 μπορεί να βρίσκεται μόνο στο τετράγωνο a2 ή στο τετράγωνο a9. Ομοίως, το 4 μπορεί να βρίσκεται μόνο στο τετράγωνο i2 ή στο τετράγωνο i9. Λόγω του μοτίβου X-Wing όπου τα κουτιά βρίσκονται στην ίδια σειρά (ή στήλη), εμφανίζεται ένας νέος λογικός περιορισμός: είναι προφανές ότι στη σειρά 2 το 4 μπορεί να βρίσκεται μόνο είτε στο τετράγωνο a2 είτε στο τετράγωνο i2 και δεν μπορεί να βρίσκεται σε κανένα άλλο τετράγωνο. Επομένως, το 4 αποκλείεται από το τετράγωνο c2, και το τετράγωνο c2 πρέπει να είναι 2.

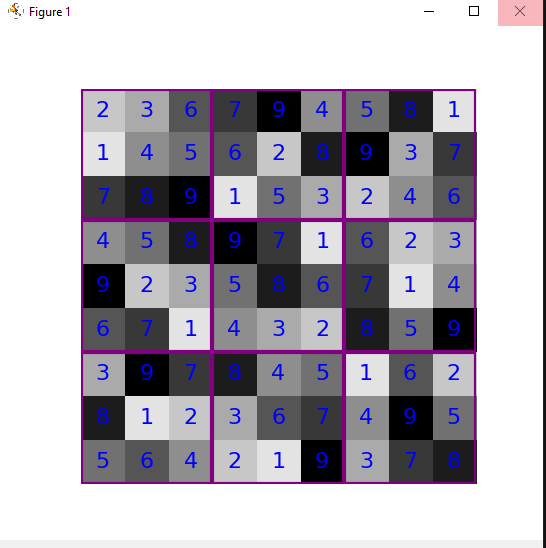


Για να υλοποιήσουμε τη μέθοδο X-Wing στη συνάρτηση apply\_x\_wing(normal\_sudoku), πρέπει να αναζητήσουμε μοτίβα X-Wing σε γραμμές και στήλες για να εξαλείψουμε τις πιθανότητες.

Το μοτίβο X-Wing εμφανίζεται όταν έχετε δύο γραμμές ή στήλες που η καθεμία περιέχει ακριβώς δύο κελιά με την ίδια υποψήφια τιμή και αυτά τα κελιά ευθυγραμμίζονται κάθετα ή οριζόντια. Εντοπίζοντας αυτό το μοτίβο, μπορείτε να εξαλείψετε την υποψήφια τιμή από άλλα κελιά στις ίδιες στήλες ή γραμμές.

Ακολουθεί ένα παράδειγμα υλοποίησης της συνάρτησης apply\_x\_wing(normal\_sudoku) για να δείξει πώς μπορείτε να βρείτε και να χρησιμοποιήσετε το μοτίβο X-Wing:







Η συνάρτηση apply\_x\_wing δέχεται ως είσοδο έναν πίνακα Sudoku, ο οποίος αναπαρίσταται ως δισδιάστατη λίστα 9x9.

Ο εξωτερικός βρόχος επαναλαμβάνει τους αριθμούς 1 έως 9, που αντιπροσωπεύουν τις πιθανές τιμές στο παζλ Sudoku.

Ο πρώτος εμφωλευμένος βρόχος ελέγχει για μοτίβα X-Wing στις γραμμές του πίνακα Sudoku. Επαναλαμβάνει πάνω από κάθε γραμμή.

Μέσα σε κάθε γραμμή, ελέγχει για κελιά που περιέχουν τον τρέχοντα αριθμό (num). Εάν βρεθεί ένα κελί, ο δείκτης στήλης του προστίθεται στη λίστα candidate\_cols. Εάν βρεθούν ακριβώς δύο στήλες (len(candidate\_cols) == 2), σημαίνει ότι υπάρχει ένα πιθανό μοτίβο X-Wing. Οι δείκτες των δύο στηλών αποθηκεύονται στα col1 και col2.

Στη συνέχεια, ο κώδικας ελέγχει αν οι ίδιες δύο στήλες εμφανίζονται και σε άλλη γραμμή (other\_row) με επανάληψη από row + 1 έως 9. Εάν οι ίδιες δύο στήλες βρεθούν σε άλλη γραμμή και τα κελιά αυτά περιέχουν επίσης τον αριθμό num, αυτό συνεπάγεται ένα μοτίβο X-Wing. Ο κώδικας προχωρά στην εξάλειψη του num από τα άλλα κελιά στις ίδιες στήλες.

Ο δεύτερος εμφωλευμένος βρόχος ελέγχει για μοτίβα X-Wing στις στήλες του πίνακα Sudoku. Επαναλαμβάνει κάθε στήλη.

Μέσα σε κάθε στήλη, ελέγχει για κελιά που περιέχουν τον τρέχοντα αριθμό (num). Εάν βρεθεί ένα κελί, ο δείκτης γραμμής του προστίθεται στη λίστα candidate\_rows.

Εάν βρεθούν ακριβώς δύο γραμμές (len(candidate\_rows) == 2), σημαίνει ότι υπάρχει ένα πιθανό μοτίβο X-Wing. Οι δύο δείκτες γραμμών αποθηκεύονται στα row1 και row2. Στη συνέχεια, ο κώδικας ελέγχει αν οι ίδιες δύο γραμμές εμφανίζονται και σε άλλη στήλη (other\_col) με επανάληψη από col + 1 έως 9.

Εάν οι ίδιες δύο

Στους προηγούμενους κώδικες χρησιμοποιήσαμε την βιβλιοθήκη PuLP της Python. Η συγκεκριμένη βιβλιοθήκη έχει συγκεριμένους επιλυτές όπου ανάλογα με το πρόβλημα επιλέγει και διαφορετικό επιλυτή. (Συνήθως τον πιο γρήγορο για το κάθε πρόβλημα). Οι επιλυτές έχουν ως εξής:

* COIN-OR (CBC): COIN-OR (Coin-Or Open Solver Interface) είναι ένας επιλυτής γραμμικού προγραμματισμού ανοικτού κώδικα. Η PuLP μπορεί να διασυνδεθεί με τον CBC (Coin Branch and Cut) για την επίλυση προβλημάτων LP.
* GLPK:Η PuLP μπορεί να διασυνδεθεί με το GNU Linear Programming Kit (GLPK), μια βιβλιοθήκη γραμμικού προγραμματισμού ανοικτού κώδικα, για την επίλυση προβλημάτων LP.
* CPLEX: Η PuLP μπορεί να διασυνδεθεί με το CPLEX της IBM, έναν ισχυρό εμπορικό επιλυτή βελτιστοποίησης που χρησιμοποιείται ευρέως για γραμμικό προγραμματισμό, μεικτό ακέραιο προγραμματισμό και τετραγωνικό προγραμματισμό.
* Gurobi: Η PuLP μπορεί να διασυνδεθεί με το Gurobi, έναν εμπορικό επιλυτή βελτιστοποίησης που είναι γνωστός για τις δυνατότητες υψηλής απόδοσης στον γραμμικό και τον μεικτό ακέραιο προγραμματισμό.
* XPRESS: Η PuLP μπορεί να διασυνδεθεί με το FICO Xpress, έναν εμπορικό επιλυτή βελτιστοποίησης με υποστήριξη γραμμικού προγραμματισμού, μικτού ακέραιου προγραμματισμού και άλλων τύπων προβλημάτων βελτιστοποίησης.

Λίγα λόγια για τους πιο σημαντικούς:

**COIN-OR (CBC):**

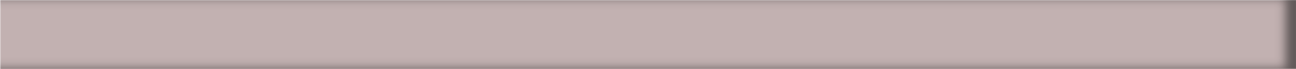
1. Διατύπωση προβλήματος: Ορίζετε το πρόβλημα γραμμικού προγραμματισμού χρησιμοποιώντας το PuLP, καθορίζοντας την αντικειμενική συνάρτηση, τους περιορισμούς και τις μεταβλητές.
2. Προεπεξεργασία: Η CBC απλοποιεί το πρόβλημα αφαιρώντας πλεονασμούς και εντοπίζοντας αδυναμίες.
3. Αρχικοποίηση: Η CBC δημιουργεί μια αρχική λύση για το πρόβλημα.
4. Διακλάδωση και οριοθέτηση: Η CBC διερευνά συστηματικά το χώρο εφικτών λύσεων με διακλαδώσεις σε μεταβλητές και διαχωρισμό του προβλήματος σε υποπροβλήματα.
5. Επίπεδα κοπής: Η CBC αυστηροποιεί τα όρια του προβλήματος ενσωματώνοντας τεχνικές κοπής που προέρχονται από τη χαλάρωση του γραμμικού προγραμματισμού.
6. Ακεραία σκοπιμότητα: Η CBC ελέγχει αν οι πιθανές λύσεις ικανοποιούν τις ακέραιες απαιτήσεις του προβλήματος.
7. Βέλτιστη λύση: Η CBC συνεχίζει την αναζήτηση μέχρι να βρει τη βέλτιστη λύση ή να φτάσει σε ένα καθορισμένο όριο.

**GLPK:**

1. Διατύπωση προβλήματος: Ορίζετε το πρόβλημα γραμμικού προγραμματισμού χρησιμοποιώντας το PuLP, καθορίζοντας την αντικειμενική συνάρτηση, τους περιορισμούς και τις μεταβλητές.
2. Προεπεξεργασία: Το GLPK απλοποιεί το πρόβλημα αφαιρώντας τις πλεονασμούς και εντοπίζοντας τις αδυναμίες.
3. Αρχικοποίηση: Το GLPK δημιουργεί μια αρχική λύση για το πρόβλημα.
4. Μέθοδος Simplex: Το GLPK χρησιμοποιεί τη μέθοδο simplex για να βελτιώσει επαναληπτικά την τιμή της αντικειμενικής συνάρτησης έως ότου βρεθεί η βέλτιστη λύση.
5. Ανάλυση ευαισθησίας: Το GLPK μπορεί να εκτελέσει ανάλυση ευαισθησίας για να κατανοήσει τον αντίκτυπο των αλλαγών στη λύση.
6. Ακεραία σκοπιμότητα: Εάν εμπλέκονται ακέραιες μεταβλητές, το GLPK χρησιμοποιεί τη μέθοδο branch and bound για την αποτελεσματική αναζήτηση εφικτών ακέραιων λύσεων.
7. Βέλτιστη λύση: Το GLPK συνεχίζει την αναζήτηση μέχρι να βρει τη βέλτιστη λύση ή να φτάσει σε ένα καθορισμένο όριο.

**CPLEX:**

1. Διατύπωση προβλήματος: Ορισμός του προβλήματος γραμμικού προγραμματισμού με τη χρήση του PuLP.
2. Προεπεξεργασία: CPLEX απλοποιεί το πρόβλημα αφαιρώντας τις πλεονασμούς και εντοπίζοντας τις αδυναμίες.
3. Αρχικοποίηση: Η CPLEX δημιουργεί μια αρχική λύση για το πρόβλημα.
4. Φάση γραμμικού προγραμματισμού: Η CPLEX επιλύει τη χαλάρωση LP χρησιμοποιώντας τεχνικές όπως η μέθοδος simplex ή οι μέθοδοι εσωτερικού σημείου.
5. Φάση ακέραιου προγραμματισμού: Εάν υπάρχουν ακέραιες μεταβλητές, η CPLEX χρησιμοποιεί αλγορίθμους διακλάδωσης και αποκοπής για την αποτελεσματική εύρεση βέλτιστων ακέραιων λύσεων.
6. Ευρετικές μέθοδοι και μέθοδοι Primal-Dual: Η CPLEX χρησιμοποιεί ευρετικές και πρωτογενείς-διπλές μεθόδους για να καθοδηγήσει την αναζήτηση εφικτών και βέλτιστων λύσεων.
7. Βέλτιστη και εφικτή λύση: Η CPLEX συνεχίζει την αναζήτηση μέχρι να βρει τη βέλτιστη ή εφικτή λύση.
8. Βελτίωση λύσεων και μετεπεξεργασία: Η CPLEX βελτιώνει τη λύση και εκτελεί εργασίες μετεπεξεργασίας, όπως ανάλυση ευαισθησίας και δημιουργία αναφορών.
9. Ανάκτηση λύσης: Η CPLEX επιστρέφει τη λύση στο PuLP για ανάλυση.



ΣΥΓΚΡΙΣΗ ΜΕΘΟΔΩΝ

1. **Σύγκριση Μεθόδων**

Η επιλογή του αλγορίθμου για την επίλυση ενός προβλήματος εξαρτάται από διάφορους παράγοντες, όπως το μέγεθος του προβλήματος, ο διαθέσιμος χρόνος και οι πόροι, η επιθυμητή ακρίβεια της λύσης και οι περιορισμοί του προβλήματος.

Γενικά, ο γραμμικός προγραμματισμός είναι κατάλληλος για προβλήματα βελτιστοποίησης που περιλαμβάνουν γραμμικές σχέσεις μεταξύ μεταβλητών και περιορισμών και όπου ο στόχος είναι η μεγιστοποίηση ή η ελαχιστοποίηση μιας αντικειμενικής συνάρτησης. Χρησιμοποιείται συχνά για την επίλυση προβλημάτων μεγάλης κλίμακας, όπου άλλες μέθοδοι όπως η brute force ή το backtracking μπορεί να είναι υπολογιστικά ανέφικτες.

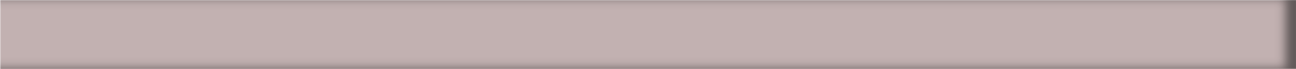
Από την άλλη πλευρά, η οπισθοδρόμηση χρησιμοποιείται συχνά για συνδυαστικά προβλήματα, όπου ο χώρος λύσεων είναι ένα πεπερασμένο σύνολο πιθανών λύσεων και ο στόχος είναι η εύρεση μιας έγκυρης λύσης. Μπορεί να είναι μια ισχυρή τεχνική για την επίλυση προβλημάτων όπως το Sudoku, αλλά μπορεί να είναι λιγότερο αποτελεσματική από τον γραμμικό προγραμματισμό για μεγαλύτερα προβλήματα ή προβλήματα που απαιτούν βελτιστοποίηση.

Για την επίλυση παζλ Sudoku, που συνήθως περιλαμβάνουν ένα πλέγμα 9x9, και οι τρεις λύτες (CBC, GLPK και CPLEX) είναι ικανοί να βρίσκουν λύσεις. Ωστόσο, δεδομένης της απλότητας και του μικρού μεγέθους των παζλ Sudoku, η διαφορά απόδοσης μεταξύ αυτών των επιλυτών είναι μηδενική.

Όσον αφορά την ευκολία χρήσης και τη διαθεσιμότητα, οι CBC και GLPK είναι επιλυτές ανοικτού κώδικα που εγκαθίστανται και ρυθμίζονται εύκολα. Ενσωματώνονται καλά με το PuLP και έχουν απλές διεπαφές. Αν η απόδοση είναι κρίσιμος παράγοντας ή αν σκοπεύετε να λύσετε παζλ Sudoku μεγάλης κλίμακας, ο CPLEX μπορεί να προσφέρει πλεονεκτήματα. Είναι ένας εμπορικός επιλυτής υψηλής απόδοσης που έχει σχεδιαστεί ειδικά για να χειρίζεται αποτελεσματικά πολύπλοκα προβλήματα βελτιστοποίησης.

Συνολικά, για την επίλυση παζλ Sudoku τυπικού μεγέθους, ο CBC ή ο GLPK θα ήταν κατάλληλες επιλογές λόγω της απλότητας, της ευκολίας χρήσης και της διαθεσιμότητάς τους ως επιλυτές ανοικτού κώδικα. Ωστόσο, αν προβλέπετε μεγαλύτερα ή πιο σύνθετα παζλ Sudoku ή αν χρειάζεστε προηγμένες δυνατότητες βελτιστοποίησης, ο CPLEX θα ήταν μια προτιμώμενη επιλογή για την επεκτασιμότητά του και τα χαρακτηριστικά υψηλής απόδοσης.

Συνοψίζοντας, η επιλογή του αλγορίθμου εξαρτάται από το συγκεκριμένο πρόβλημα και τους περιορισμούς του και είναι σημαντικό να εξετάζονται τα πλεονεκτήματα και οι περιορισμοί των διαφόρων μεθόδων προτού αποφασιστεί ποια θα χρησιμοποιηθεί.



ΒΙΒΛΙΟΓΡΑΦΙΑ

1. *Bartlett A.D., Chartier T.P., Langville A.N., Rankin T. An Integer Programming Model for the Sudoku Problem, J, Online Math. Applicat., vol. 8, 2008*
2. *R. J. Vanderbei, "Linear programming: foundations and extensions," International Series in Operations Research & Management Science, vol. 116, Springer, 2014, pp. 213-230.*
3. *T. K. Ralphs, L. E. Trotter Jr., and J. P. Yunes, "A mixed integer programming model for the Sudoku problem," INFORMS Journal on Computing, vol. 20, no. 3, pp. 390-398, 2008.*
4. *S. Gassner and D. Ullman, "Solving Sudoku puzzles with integer programming," INFORMS Transactions on Education, vol. 8, no. 1, pp. 1-9, 2007.*
5. *M. J. Kim and H. J. Kim, "Sudoku puzzle generation using integer linear programming," IEICE Transactions on Information and Systems, vol. E96-D, no. 7, pp. 1515-1518, 2013.*
6. *Y. Cheon and H. Shin, "An efficient algorithm for Sudoku puzzles based on linear programming," Journal of Computational and Applied Mathematics, vol. 303, pp. 29-38, 2016.*
7. *https://www.sudokuoftheday.com/techniques/x-wings#:~:text=X%2DWings%20are%20when%20there,two%20positions%20for%20a%20number.&text=Once%20you've%20satisfied%20yourself,in%20rows%204%20and%209.*