Constraint Satisfaction Problems: Sudoku Solver

Csp, არის ამოცანების ტიპი, რომელთა სთეითებსაც ახასიათებთ სასრული რაოდენობის წინასწარ მითითებული შეზღუდვები. ანუ ამ ამოცანების ამომხსნელის მიზანია იუზერის მიერ მითითებული შეზღუდვების დამაკმაყოფილებელი პასუხი იპოვოს. ეს ამოცანები ფორმალურად განისაზღვრება სასრული რაოდენობის ცვლადებითა და შეზღუდვებით. სადაც, თითოეულ ცვლადს გააჩნია დომეინი, ანუ შესაძლო მნიშვნელობების სია, რომელიც ამ ცვლადზე არსებული შეზღუდვების დაკმაყოფილებით შეგვიძლია ავირჩიოთ.

ასეთი ამოცანების მაგალითია სოდოკუ. მისი ცვლადებია თითოეული უჯრის კოორდინატები. შეზღუდვებია იმ რიცხვების ჩაწერის შეუძლებლობა, რომელიც შესაბამის სვეტში, სტრიქონში ან კვადრატში წერია. დომეინი კი შესაბამისად ის რიცხვები გამოდის, შესაძლო {1,2,3,4,5,6,7,8,9}-დან, რომლებიც კონკრეტული უჯრის შესაბამის სვეტში/სტრიქონში/კვადრატში არ გვხვდება. ამოცანის მიზანია შევავსოთ ყველა უჯრა ისე, რომ ყველა მათგანისთვის იყოს ეს შეზღუდვები დაკმაყოფილებული.

Backtracking Search

Csp-ების ამოხსნის ალგორითმები რეკურსიული სახისაა და ხშირად იყენებენ ბექთრექინგს. ბექთრექინგი არის, უხეშად რომ ვთქვათ, ცვლილებების წაშლა როდესაც შეცდომას ვაწყდებით, რაც ამ შემთხვევაში შეზღუდვის დარღვევა გამოდის. ბექთრექინგიანი რეკურსიის გამოყენება ბუნებრივია, სწორ შედეგამდე მიგვიყვანს, მაგრამ ბევრ სთეითიანი ამოცანისთვის სადაც ბევრი შემთხვევის განხილვა გვიწევს, ძალიან დიდ დროს უნდება.

Forward Checking (FC)

შემთხვევების შემცირებისთვის ერთ-ერთი პოპულარული/გამოსადეგი ალგორითმია forward-checking. მისი არსი შემდეგში მდგომარეობს: დამატებით ბექთრქინგიანი რეკურსიისა, ალგორითმი თითოეულ ცვლადში შესაძლო მნიშვნელობების ჩაწერის განხილვისას ამ მნიშვნელობას ყველა სხვა ცვლადის დომეინიდან შლის, რომლის შეზღუდვასაც ირღვევა. ანუ, სუდოკუს შემთხვევაში, თუ x,y უჯრაში m რიცხვი ჩავწერეთ, მას x,y-ის შეაბამისი სვეტის, სტრიქონის და კვადრატის ყველა სხვა უჯრის დომეინიდან წავშლით (სადაც m იყო დომეინში საერთოდ). ხოლო ამ უჯრაში დომეინად მხოლოდ m-ს დავტოვებთ. საგლისხმოა, რომ ამ ალგორითმის ბექთრექინგი ითვალისწინებს იმ დომეინებში, საიდანაც m ამოვშალეთ, მის დაბრუნებასაც.

Arc Consistency(AC)

Csp-ების კიდევ ერთი ცნობილი ალგორითმია arc consistency. მისი არსი მდგომარეობს იმაში, რომ ყოველი ცვლადისადმი მნიშვნელობის მინიჭებისას, მის შესაბამის arc-ში უნდა მოიძებნებოდეს თითო მაინც შესაძლებელი მნიშვნელობა, რომელიც კონსისტენტურობას არ დაარღვევს. მაგალითად, ორი “მეზობელი” (სუდოკუს შემთხვევაში უჯრისათვის მეზობლებია მისი შესაბამისი სვეტის/სტრიქონის/კვადრატის უჯრა) ცვლადი თუ გვაქვს და პირველში 9-ის ჩაწერა, მეორეს დომეინს აცარიელებს, გამოდის რომ კონსისტენტურობა ირღვევა და ამ მნიშვნელობას აღარ ჩავწერთ.

სუდოკუს შემთხვევაში forward checking უფრო გვადგება ვიდრე arc consistency, რადგან ეს უკანასკნელი ბევრ დროსა და რესურსს მოითხოვს, რადგან თითოეული მეზობელი ცვლადის დომეინში შედის ღრმად, თითოეულზე ამოწმებს კონსისტენტურობას და ა.შ.

გარდა თავად ამ ალგორითმებისა, არსებობს კიდევ დამხმარე ევრისტიკები, რომლებიც ამ ალგორითმების მიერ გადაყოლილი ცვლადებისა თუ მნიშვნელობების თანმიმდევრობას განსაზღვრავენ და ამის ხარჯზე საგრძნობლად აჩქარებენ მათ.

MRV

იშიფრება და ითარგმნება, როგორც მინიმალური დარჩენილი მნიშვნელობები. და გულისხმობს იმ ცვლადის არჩევას, რომელსაც ყველაზე ცოტა შესაძლო მნიშვნელობა აქვს დარჩენილი, ანუ რომლის დომენიც ყველაზე პატარაა. ეს ევრისტიკა ძალიან გამოსადეგია სუდოკუში, იმიტომ, რომ თითოეული უჯრის შევსება ბევრი სხვა ცვლადის დომეინს ამცირებს და ამოცანის ამოხსნა დადის ყველა დომეინის გაერთებამდე შემცირებაზე. ამიტომ ისეთი დომეინით დაწყება რომელიც ყველაზე პატარაა და ყოველთვის პირველად ასეთი დომეინების არჩევა ძალიან ეფექტურია. ინტუიტირუად ყველა ადამიანიც მსგავსად უდგება სუდოკუს ამოცანის გადაჭრას.

Degree

დიგრი ანუ ხარისხის ევრისტიკა არის mrv-ს tie breaker, ანუ იგი არჩევს მინიმალურს ორ ერთნაირი სიგრძის დომეინიან ცვლადს შორის. ამას აკეთებს მეორე პარამეტრით, გარდა სიგრძისა, რომელიცაა ხარისხი ანუ ამ ცვლადისთვის წიბოების რაოდენობა. დომეინების ენაზე: რომელ ცვლადს აქვს ყველაზე მეტი მიუნიჭებელი ცვლადი თავის მოქმედების არეალში, ანუ იმ ადგილებზე, რომლებზეც ამაში მნიშვნელობების ჩაწერა მქომედებს. სუდოკუს ამოცანისთვის ეს ევრისტიკა დადებითზე მეტად უარყოფითს გვაძლევს, რადგან მისი განხორციელება დროის მოგებით ძალიან რთულია. მე პირადად გამოვიყენე პითონის ოპერატორის კლასი, მაგრამ სხვა გზითაც რომ

გაკეთდეს, ორი პარამეტრით სორტირება ძალიან ბევრ რესურსს მოითხოვს, როცა, რადგან სუდოკუ მხოლოდ 81 ცვლადისგან შედგება არც ისე ბევრი tie გვვდება მრვ-ის გამოყენებისას, რომ სიჩქარეს დიდად აფუჭებდეს.

LCV

იშიფრება და ითარგმნება, როგორც ყველაზე ნაკლებად შემზღუდველი მნიშვნელობა. იგი, კონკრეტული ცვლადისთვის, მნიშვნელობებს ალაგებს და გადაუყვება იმის მიხედვით თუ რომელი უფრო ნაკლებ მნიშვნელობას ამოშლის მეზობელი (ანუ სვეტში/სტრიქონში/კვადრატში მყოფი) უჯრების დომეინებიდან. ეს ევრისტიკა ცდილობს მაქსიმალური გასაქანი მისცეს ალგორითმს ყველა სხვა ცვლადის მნიშვნელობების არჩევისას. რაც სუდოკუსთვის ძალიან მოუხერხებელია, რადგან ჩვენი მიზანია რომ რაც შეიძლება მეტი უჯრის შემთხვევაში შევამციროთ დომეინები ერთამდე, ანუ რაც შეიძლება მაქსიმალურად შევზღუდოთ რიცხვების არჩევის შესაძლო ვარიანტები. სუდოკუსგან განსხვავებით, მაგალითად, გრაფის შეღებვის ამოცანაში, სადაც მხოლოდ 3 ფერი ფიგურირებს, ჩვენთვის ის უფრო მოსახერხებელი და სწრაფია რომ ეს ფერები არ ამოგვეწუროს, რადგან ასე ბექთრექინგი უფრო ნაკლებად გვიწევს.

ევრისტიკების პერფორმანსი:

	MRV	MRV + Degree	MRV + LCV
1	6,4	128,7	10,7
2	6,7	130,8	10,8
3	6,5	129,8	10,85
Avg	6,533	129,7666667	10,783333

მოცემულია სამივე ევრისტიკაზე რენდომ სამი მცდელობის შედეგად პროგრამის მიერ დახარჯული წამები და ბოლო სტრიქონზე: შესაბამისი საშუალო პერფორმანსი.

როგორც ვხედავთ, საუკეთესო შედეგს mrv გვაძლევს. მხოლოდ lcv-ს გასვება იმდენად ანელებს პროგრამას, რომ მის გაშვებას საერთოდ არ აქვს აზრი. ეს შედეგები ასაბუთებს ზემოთ მოცემულ მსჯელობას, იმის შესახებ თუ რატომ სჯობს mrv lcv-ს ამ კონკრეტულ შემთხვევაში, მითუმეტეს, რომ ეს უკანასკნელი პირველთან ერთადაც კი აუარესებს პერფორმანსს. დიგრის შემთხვევაში კი ძირითადი პრობლემა იმპლემენტაციაა, მაგრამ მაინც ზედმეტი რესურსია ასეთ პატარა მასშტაბის ამოცანაზე მისი დაწერა.

არქიტექტურა:

ჩემი პროგრამა მოიცავს მეინ მეთოდს, სადაც ხდება, ფაილის წაკითხვა და დამხმარე მეთოდით 3 განზომილებიან ლისტში ჩაწერა სუდოკუს ცხრილების, ასევე სხვა ძირითადი სტრუქტურების აწყობა და შემოღება. ასევე მოიცავს დამხმარე მეთოდებს, რომლებიც ამ ოპერაციებს ამარტივებენ. მაგალითად: დომეინის ამოღება, მეზობელი ნოუდების ამოღება, ცარიელი ანუ მიუნიჭებელი ცვლადების სიის ამოღება. შემდეგ კი თითოეული სუდოკუსთვის იძახება ფორვარდ ჩექინგ ალგორითმის მეთოდი. ჩემი ძირითადი ამოხსნა მოიცავს ამ ალგორითმისა და mrv ევრისტიკის კომბინაციას. ქვემოთ სწორედ მათი ფსევდოკოდებია მოცემული. მაგრამ ასევე ფოლდერში წააწყდებით ალტერნატიულ ფაილებს, სადაც MRV + degree და MRV + LCV ევრისტიკებით დაწერილი ფორვარ ჩექინგია. რომელთაგან ორივე ბევრად უფრო ნელა მუშაობს და ფაილები უბრალოდ მუშაობის აღწერის პროცესის ნაწილია.

ძირითადი ნაწილის ფსევდოკოდი:

```
MRV (unassigned, domains):
        for each unassigned variable:
                get its domain length
        return minimum variable by its domain length


CSP_FC (table, unassigned, domains):
        if table is completely filled in:
                return table
        var = next variable from unassigned values according to mrv heuristic
        remove var from unassigned
        For each val in domains(var)
                var = val
                domains = update Domains in var's area according to val, constraints
                save domain(var) and then make it [var]
                save the list of updated domain coordinates
                for each other unassigned value if neither of their domains are empty:
                        result = CSP_FC (table, unassigned, domains)
                        if result != none:
                                return result
                #backtrack
                var = empty
                domains = bring m back to Domains according to the coordinates
                reset domain(var) to its previous, saved meaning
        if var is still empty: put it back to unassigned
        return none
```

დასკვნა: სუდოკუს ამოხსნისათვის საუკეთესო ალგორითმია ფორვარ ჩექინგი + ბექთრექინგი და mrv ევრისტიკა. Lcv ანელებს ამ ალგორითმს, ხოლო mrv-ზე degree-ს დამატება ბევრ რესურსს მოითხოვს, დამოუკიდებლად კი mrv უფრო ეფექტური და საჭიროა.