2) Solving Discrete Constraint Satisfaction Problems (CSP) *Mahin*

First Draft

Name: Hyundoo Jeong
PSID:2212332

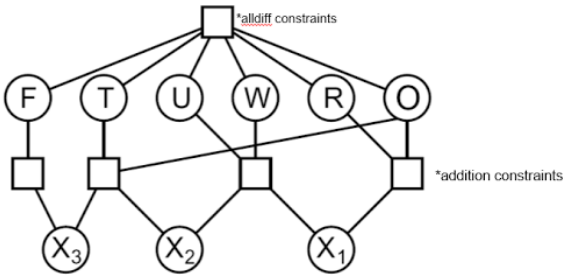

Fig. 5: Constraint Hypergraph for a Letter Equation CSP

Write a program which finds solution to the following 3 hierarchically organized[1] constraint satisfaction problems, involving 13 variables {A, B, C, …, M} which can take integer values in {1, …, 120}; solutions can use the same integer for different variables.

a. Problem A: Find a solution to the constraint satisfaction problem involving the six variables A, B, C, D, E and F and constraints C1, …, C5:
   - (C1) A=B**2– C**2
   - (C2) E+F>B
   - (C3) D=B**2-3*A
   - (C4) (B-C)**2=E*F*C-1861
   - (C5) C+D+E+F<120

   Answers:
   Solution is A = 15, B =8, C = 7, D = 19, E = 7, F = 38
   Number of variables assignments: 3393

b. Problem B: Find a solution to the constraint satisfaction problem involving ten variables A, …, J which satisfy constrains C1, …, C12:
   - (C6) (G+**I**)**3= (H-A-1)**2
   - (C7) B*E*F=H*B-200
   - (C8) (C+**I**)**2=B*E*(G+1)
   - (C9) G+**I**<E
   - (C10) D+H>180
   - (C11) J < H-C-G
   - (C12) J > B*G+D+E+G

   Answer: Solution is A=111,B=20,C=17,D=67,E=10,F=11,G=1
   H=120,I=3,J=99
   Number of variables assignments: 160226019

[1] A solution of the higher numbered problem also represents a solution of the lower numbered problem!

c. Problem C: Find a solution to the constraint satisfaction problem involving 13 variables A, …, M which satisfy constrains C1, ..., C17:
   - (C13) K*L*M=B*(B+5)
   - (C14) F**3=K*K*M*M*10 + 331
   - (C15) H*M*M=J*K-20
   - (C16) J+L=**I***L
   - (C17) A+D+M=B*(F-2)

   Answer:
   Solution C is A =111, B = 20, C =17 , D= 67, E = 10, F=11,G=1,H=120, I = 3, J = 100 K =5, L = 50, M = 2
   Number of variables assignments: 160226020

Remark: In the above equations the letter 'I' was put into bold face to avoid being mistaken as the number 1. Moreover, the letter 'J' looks somewhat similar to the letter 'I' but to better distinguish the two letters 'J' is never in bold face.

Your program should contain a counter **nva** ("number of variable assignments) that counts the number of times an initial integer value is assigned to a variable or the assigned integer to the particular variable is changed; in addition to outputting the solution to the CSV also report the value of this variable at the end of the run, and develop an interface to call your program for CSP Problems A, B, or C. Your program should return a solution or "no solution exists" and the value of nva after the program terminates. Moreover, terminate the search as soon as you found a solution—do not search for additional solutions.

Submit a report which

- Gives a brief description of the strategy you used to solve the CSP
- Provides Pseudo Code of your CSP solver
- Explains the Pseudo Code in a paragraph or two
- Describes strategies (if you employed any) you employed to reduce the runtime of your program, measured by the final value of the variable nva.
- Conducting a mathematical pre-analysis to eliminate variables, to obtain additional '<' or '>' constraints to reduce search complexity or developing other problem complexity reduction strategies based on such a pre-analysis, helps to create an efficient solution. Describe the results of the pre-analysis you conducted, and how the results of this pre-analysis were used for reducing the search complexity.
- Explain how your program takes advantage of the hierarchical structure[2] of the three CSP problems.

[2] If your approach uses solutions of a lower problem to solve the higher problem, e.g. uses solutions of problem A to solve problem B then the proper value for the variable nva should be computed by adding the cost of creating the solutions for A and the cost of finding a single solution for B based on the solutions obtained for A.

- Developing a generic program in the sense that its code could be reused to solve other constraint satisfaction problems which have a similar structure, but different constraints is expected. Include a paragraph presenting evidence why your program has this property and what you did to make your program 'generic'.,

Moreover, submit the source code for the implementation in a separate file and instructions on how to run your code in a Readme File. Attach the Readme file as an appendix to your report.

Notes on grading:

- Sophisticated approaches that lead to lower complexities in solving the respective CSPs—measured by the final value of the variable nva—will get up to 30% higher scores compared to programs that use brute force approaches.
- Severe penalties will be assessed if the value of the variable nva is not properly computed.

Report

CSP Solver Strategy: The strategy involves a brute-force search through the possible variable assignments while adhering to the specified constraints. For each problem, nested loops iterate through the variables, checking constraints at each level and returning the first valid solution found.

Pseudo Code:

```
def solvecsp():
        for A~M variable loop which couldn't express other constraints
        and most inner loop declare what is nva
        if check: domain is right a~m between (1,120)
                print(a~m, nva)
        else
                print("No solution")
```

Explanation: The pseudo code follows a straightforward approach, nesting loops for each variable and checking constraints within each level. So, it literally iterate what is value matches conditions

Reducing time complexity:

1. Variable domain reduction: Restricting domain of variables based on constraints narrows down possibilities.
2. Constraint ordering: Arranging constraints to minimize backtracking and faster performance than just ascending order.
3. Reducing For loop: if you just apply condition, you have to use much for loop which it made time take a lot, so reducing time complexity is less using for loop, I did some constraints expressed other constraint and set up domain, and eventually it made my program less for loop.

Mathematically Pre-Analysis: To eliminate some variable, you can express one variable with including other variable. For example, given variable A~F, you can use a = b * c which is already 2 variable. Like this, one variable composed multiple variable's operation. Also, some of conditions are easily getting to get more "><". For example, A= B**2 – C**2 In these cases, we know b is greater than c, and it is helping to build right order for loop.

Pros of Hierarchical structure: The pros of hierarchical structure is all cases inherit, and even though the cases are more complicated, you can use previous cases so it would easily build complicated program following information. You can solve each program gradually.

General Program Development: the program's generality is a result of deliberate design choices such as modular function design, parameterization, dynamic constraint integration, separation of concerns, and generic variable handling. These features collectively empower users to reuse the solver for a broad spectrum of CSPs by facilitating easy customization and extension.

Actual source code(Read me)

Problem A:

```python
def solveCSP():
    nva = 0
    for b in range(2,121):
        nva += 1
        for c in range(1,b):                # from C1, c needs to be less than b
            nva += 1
            for e in range(1,121):
                nva += 1
                if ((pow((b - c),2) + 1861) / (e * c)).is_integer():
                    f = int((pow((b - c),2) + 1861) / (e * c))
                    nva += 1
                    if (e + f) > b and f < (120 - c - e + (2*b*b - 3*c*c)) and 1 <= f <= 120
                        a = b*b - c*c
                        d = -2*b*b + 3*c*c
                        nva += 2
                        if 1 <= a <= 120 and 1 <= d <= 120:
                            return a,b,c,d,e,f,nva
    return 0,0,0,0,0,0,nva


print(solveCSP())
```

Problem B:

practice.py > solveCSP

```python
import math
def solveCSP():
    nva = 0
    for b in range(1, 121):
        for c in range(1, b):  # C1: c needs to be less than b
            for e in range(1, 121):
                for g in range(1, e): #C9: g needs to be less than e
                    for j in range(1, 121):
                        nva +=1
                        a = b * b - c * c  # C1
                        f = int(((b - c) ** 2) + 1861) // (e * c)  # C4

                        # Move assignments right after they can be assigned
                        i = int(math.sqrt(b * e * (g + 1)) - c)
                        h = int(e * f + (200 / b))
                        d = -2 * b * b + 3 * c * c  # C3

                        # Check the domain of the newly assigned variables
                        if (
                            1 <= a <= 120
                            and 1 <= d <= 120
                            and 1 <= f <= 120
                            and 1 <= h <= 120
                            and 1 <= i <= 120
                            and (g + i) ** 3 == (h - a - 1) ** 2  # C6
                            and b * e * f == h * b - 200  # C7
                            and (c + i) ** 2 == b * e * (g + 1)  # C8
                            and g + i < e  # C9
                            and d + h > 180  # C10
                            and j < h - c - g  # C11
                            and j > b * g + d + e + g  # C12
                        ):
                            return a, b, c, d, e, f, g, h, i, j, nva

    return 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, nva
```

Problem C:

practice.py > solveCSP

```python
import math

def solveCSP():
    nva = 0
    for b in range(1, 121):
        for c in range(1, b):  # C1: c needs to be less than b
            for e in range(1, 121):
                for g in range(1, e):  # C9: g needs to be less than e
                    for j in range(1, 121):
                        f = int(((b - c) ** 2) + 1861) // (e * c)
                        nva += 1
                        if (e + f) > b and f < (120 - c - e + (2 * b * b - 3 * c * c)) and 1 <= f <= 120:
                            a = b * b - c * c
                            d = -2 * b * b + 3 * c * c
                            h = int(e * f + (200 / b))
                            m = b * (f - 2) - (a + d)  # C17
                            i = int(math.sqrt(b * e * (g + 1)) - c)
                            if 1 <= a <= 120 and 1 <= d <= 120 and 1 <= m <= 120 and 1 <= i <= 120 and 1 <= h <= 120:
                                if (g + i) ** 3 == (h - a - 1) ** 2 and g + i < e and d + h > 180 and j < h - c - g and j > b * g + d + e + g:
                                    # Move assignments right after they can be assigned
                                    k = int((h * m * m + 20) / j)  # c15
                                    l = int(b * (b + 5) / (k * m)) #c13
                                    # C4
                                    # Check the domain of the newly assigned variables
                                    if f**3 == k*k*m*m*10 + 331 and j + l == i * l:
                                        return a, b, c, d, e, f, g, h, i, j, k, l, m, nva

    return 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, nva

print(solveCSP())
```