CPSC 322 Assignment 1

Jose Abraham Torres Juarez - 79507828
Gustavo Martin - 62580121

1 Question 1 (21points): Allocating Developments Problem

(a) Represent this problem as a CSP

(a) **Variables:**

$$\{hc, bh, ra, gd\}$$

(b) **Domains:**

$$\{(0,0),(0,1),(0,2),(1,0),(1,1),(1,2),(2,0),(2,1),(2,2)\}$$

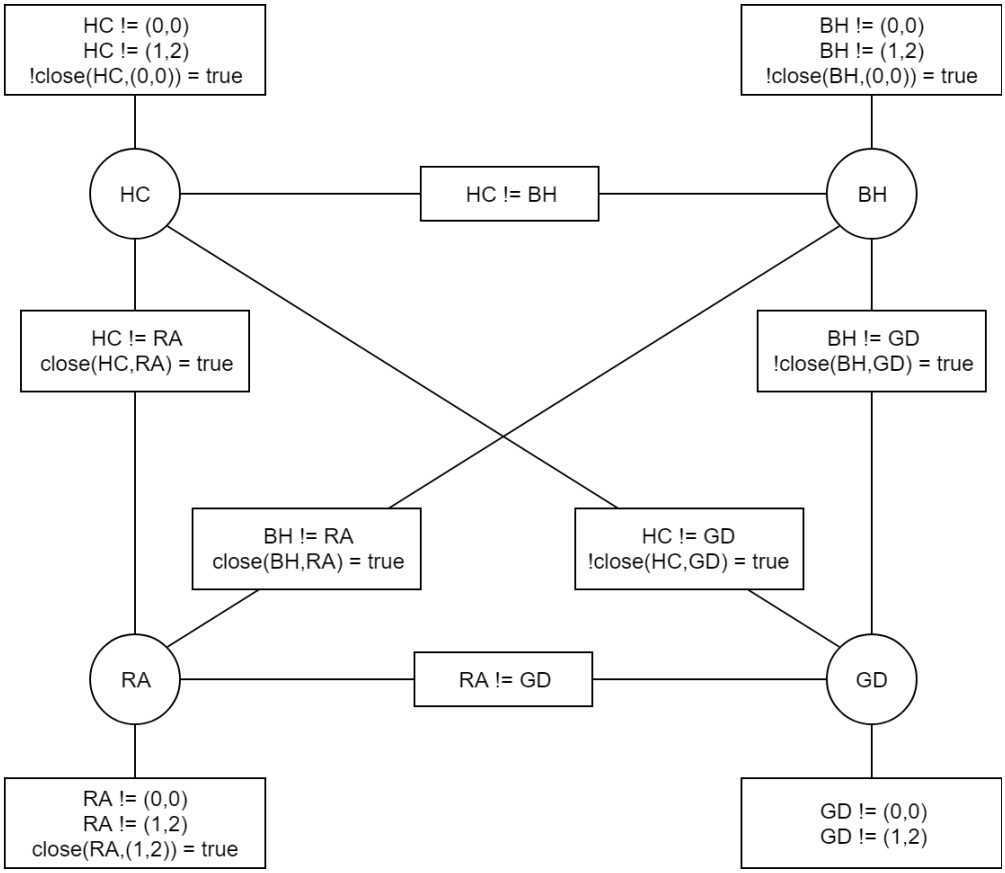
(c) **Constraints:**

$$\{hc \neq ra, hc \neq dg, hc \neq bh, bh \neq ra, bh \neq gd, ra \neq gd,$$
$$hc \neq (0,0), bh \neq (0,0), ra \neq (0,0), gd \neq (0,0),$$
$$hc \neq (1,2), bh \neq (1,2), ra \neq (1,2), gd \neq (1,2),$$
$$!Close(hc,(0,0)), !Close(bh,(0,0)), !Close(hc,gd), !Close(bh,gd)$$
$$Close(ra,(1,2)), Close(hc,ra), Close(bh,ra)\}$$

For our constraint definitions we use the following functions:

Function *Close* (A, B)
// Check if $||A - B||_1 == 1$, the Manhattan distance must be 1
return $(|A[0] - B[0]| + |A[1] - B[1]|) == 1$

(b) Draw a constraing graph for this problem


HC != (0,0)
HC != (1,2)
!close(HC,(0,0)) = true
BH != (0,0)
BH != (1,2)
!close(BH,(0,0)) = true
HC
HC != BH
BH
HC != RA
close(HC,RA) = true
BH != GD
!close(BH,GD) = true
BH != RA
close(BH,RA) = true
HC != GD
!close(HC,GD) = true
RA
RA != GD
GD
RA != (0,0)
RA != (1,2)
close(RA,(1,2)) = true
GD != (0,0)
GD != (1,2)

2 CSP - Search

(a) Show how search can be used to solve this problem, using the variable ordering A, B, C, D, E, F, G, H.
See **Appendix A** for the implementation of `CSP`.

(b) Is it possible to generate a smaller tree?

(c) Explain why you expect the heuristic in part (b) to be good

3 CSP - Arc Consistency

(a) Show how arc consistency can be used to solve the scheduling problem in Question 2

(b) Use domain splitting to solve this problem

(c) Constraint satisfaction problems can become extremely large and complex.

4 CSP - Stochastic Local Search

(a) For one particular run, make SLS select any variable that is involved in an unsatisfied constraint, and select a value that results in the minimum number of unsatisfied constraints. Report the initialized values. At each step, report which variable is changed, its new value, and the resulting number of unsatisfied arcs. (You only need to do this for 5 steps).

(b) Using batch runs, compare and explain the result of the following settings:

(i) Select a variable involved in the maximum number of unsatisfied constraints, and the best value.

(ii) Select any variable that is involved in unsatisfied constraints, and the best value.

(iii) Select a variable at random, and the best value.

(iv) A probabilistic mix of i. and ii.Try a few probabilities and report on the best one you find

(c) Pick one of the methods(ii)-(iv)from part (b) and run it repeatedly (5 runs should be enough, but you may do more runs if you wish). State which method you chose. Give a screenshot of the results; then think about, describe and explain whatyou observe.

(d) How important is it to choose the value that results in the fewest unsatisfied constraints as opposed to choosing a value at random? Justify your answer with evidence.

(e) For the best variable/best value method from part(b), allow random resets (for example, after 50 steps). Give a screenshot comparing the results to what happens if random resets are not allowed, and explain how this affects the performance of the algorithm, and why.

A Appendix: CSP as Search implementation

```
class Node:
    def __init__(self, name, value=None, depth = 0, values = []):
        self.adjacents = []
        self.name = name
        self.value = value
        self.depth = depth
        self.values = values.copy()
        if depth > 0:
            self.values[depth-1] = value

    def description(self):
        if self.depth == 0:
            return ""
        return "{}={}".format(self.name,self.value)

    def link(self, node):
        self.adjacents.append(node)


def generateTree(useHeuristic = False, file = None):
    Domain = [1,2,3,4]
    varNames = ['A','B','C','D','E','F','G','H']
    mapVarNames = {'A':0,'B':1,'C':2,'D':3,'E':4,'F':5,'G':6,'H':7}
    varN = len(varNames)
    constraints = {
```

```
    'AG':lambda A, G : A >= G,
    'AH':lambda A, H : A < H,
    'FB':lambda F, B : abs(F-B) == 1,
    'GH':lambda G, H : G < H,
    'GC':lambda G, C : abs(G-C) == 1,
    'HC':lambda H, C : abs(H-C) % 2 == 0,
    'HD':lambda H, D : H != D,
    'DG':lambda D, G : D >= G,
    'DC':lambda D, C : D != C,
    'EC':lambda E, C : E != C,
    'ED':lambda E, D : E < D-1,
    'EH':lambda E, H : E != H-2,
    'GF':lambda G, F : G != F,
    'HF':lambda H, F : H != F,
    'CF':lambda C, F : C != F,
    'DF':lambda D, F : D != F,
    'EF':lambda E, F : abs(E-F) % 2 == 1,
}
# Helper that check if constraints are valid for the provided values
def checkConstraints(values):
    for key, constraint in constraints.items():
        X = values[mapVarNames[key[0]]]
        Y = values[mapVarNames[key[1]]]
        if X == None:
            continue
        if Y == None:
            continue
        if not constraint(X,Y):
            return False
    return True

# Helper to change the order we pick the variables
def varNamesByHeuristic():
    heuristics = {}
    newNames = []
    mapNewNames = {}
    for i in range(varN):
        varName = varNames[i]
        heuristics.update({varName:0})
        for key, constraint in constraints.items():
            if varName in key:
                heuristics[varName] += 1
    i = 0
    for key in sorted(heuristics.keys(), reverse=True):
        mapNewNames.update({key: i})
        newNames.append(key)
        i += 1

    return newNames, mapNewNames

root = Node("Root", values=[None]*8)
frontier = [root]
solutions = []
fails = 0
newLine = False
if useHeuristic:
```

```
        varNames,mapVarNames = varNamesByHeuristic()
    f = None
    if file != None:
        f = open(file,"w+")

    while(len(frontier)>0):
        n = frontier.pop()
        # Print the node description (Variable=Value)
        if n.depth > 0:
            nTabs = 1
            if newLine:
                nTabs = n.depth
                newLine = False
            description = "{}{}".format("\t"*nTabs, n.description())

            if file != None:
                f.write(description)
            else:
                print(description, end="")

        # Check if the current assigned variables are valid with constraints
        if not checkConstraints(n.values):
            if file != None:
                f.write("failure\n")
            else:
                print("failure")
            fails += 1
            newLine = True
            continue

        # Check if we hit a solution
        if n.depth == varN:
            if file != None:
                f.write("solution\n")
            else:
                print("solution")
            solutions.append(n.values)
            newLine = True
            continue

        # Add next variable with all domains
        for i in range(len(Domain)):
            new = Node(varNames[n.depth], Domain[i], n.depth+1, n.values)
            n.link(new)
            frontier.append(new)
    f.close()
    return varNames, solutions, fails

if __name__ == "__main__":
    # variables, solutions, fails = generateTree(False, "../data/q2_tree.txt")
    variables, solutions, fails = generateTree(True, "../data/q2_tree_heur.txt")
    varN = len(variables)

    print("\nNumber of fails: {}".format(fails))
    print("\nNumber of Solutions: {}".format(len(solutions)))
    # Print the solutions
```

```
print("\nSolutions:\n")
for i in range(varN):
    print("\t{}".format(variables[i]),end="")
print()
for i in range(len(solutions)):
    for j in range(varN):
        print("\t{}".format(solutions[i][j]),end="")
    print()
```