**Pair Programming Equitable Participation & Honesty Affidavit**

We the undersigned promise that we have in good faith attempted to follow the principles of pair programming. Although we were free to discuss ideas with others, the implementation is our own. We have shared a common workspace and taken turns at the keyboard for the majority of the work that we are submitting. Furthermore, any non programming portions of the assignment were done independently. We recognize that should this not be the case, we will be subject to penalties as outlined in the course syllabus.

Pair Programmer 1 (print & sign your name, then date it) 

Pair Programmer 2 (print & sign your name, then date it)

*'''*

*file: backtrack.py*

*Shawn Chua and Alexander Giang*

*'''*

from csp\_lib.backtrack\_util import (first\_unassigned\_variable,

unordered\_domain\_values,

no\_inference)

# pseudocode: slide 44

def **backtracking\_search**(csp,

select\_unassigned\_variable=first\_unassigned\_variable,

order\_domain\_values=unordered\_domain\_values,

inference=no\_inference):

*"""backtracking\_search*

*Given a constraint satisfaction problem (CSP),*

*a function handle for selecting variables,*

*a function handle for selecting elements of a domain,*

*and a set of inferences, solve the CSP using backtrack search*

*"""*

# See Figure 6.5] of your book for details

def **backtrack**(assignment):

*"""Attempt to backtrack search with current assignment*

*Returns None if there is no solution. Otherwise, the*

*csp should be in a goal state.*

*"""*

# if all variables assigned, return assignment

if assignment.\_\_len\_\_() == csp.variables.\_\_len\_\_():

return assignment

var = select\_unassigned\_variable(assignment,csp)

#for each value in order domain

for value in order\_domain\_values(var, assignment, csp):

#if value is consistent with assignment (no conflicts)

if csp.nconflicts(var,value,assignment) == 0:

csp.assign(var,value,assignment) #assign value

#suppose() value for interference() & restore()

removals = csp.suppose(var,value)

#check if there's inference

if inference(csp, var, value, assignment, removals):

result = backtrack(assignment)

#if backtrack did not fail, return result

if result is not None:

return result

# either value inconsistent or further exploration failed

csp.restore(removals) #undo suppose

# restore assignment to its state at top of loop and try next value

csp.unassign(var,assignment)

return None #failure

# Call with empty assignments, variables accessed

# through dynamic scoping (variables in outer

# scope can be accessed in Python)

result = backtrack({})

assert result is None or csp.goal\_test(result)

return result

*'''*

*Constraint propagation*

*file: constraint\_prop.py*

*Shawn Chua and Alexander Giang*

*'''*

#pseudocode: slides 29 and 30

def **AC3**(csp, queue=None, removals=None):

*"""AC3 constraint propagation*

*"""*

# Hints:

# Remember that:

# csp.variables is a list of variables

# cps.neighbors[x] is the neighbors of variable x

if queue != None: #if queue isn't empty, use the queue

q = queue

else: #queue is empty

q = [] #initialize an empty queue

csp.support\_pruning() # MUST BE CALLED before starting to prune

#fill in queue with binary arcs in CSP

for x in csp.variables:

for y in csp.neighbors[x]:

q.append((x,y))

# while queue isn't empty

while q.\_\_len\_\_() != 0:

(xi,xj) = q.pop() #get binary constraint (dequeue)

if revise(csp,xi,xj,removals):

#if di = Nothing, return false

if csp.curr\_domains[xi] is None:

return False

else:

#for each xk in neighbors(xi) - xj

for xk in (csp.neighbors[xi] - {xj}):

q.append((xk,xi)) #enqueue

return True

def **revise**(csp,xi,xj,removals):

pruned = False # initially assume there's no revision

#for each x in di

for x in csp.curr\_domains[xi]:

#if y doesn't exist in dj such that constraint holds between x & y

if all(not csp.constraints(xi, x, xj, y) for y in csp.curr\_domains[xj]):

csp.prune(xi,x,removals) #delete x from di

pruned = True

return pruned

*'''*

*file: driver.py*

*Shawn Chua and Alexander Giang*

*'''*

from csp\_lib.sudoku import (Sudoku, easy1, harder1)

from constraint\_prop import AC3

from csp\_lib.backtrack\_util import mrv

from backtrack import backtracking\_search

for puzzle in [easy1]:

easy = Sudoku(easy1) # constructing the easy Sudoku problem

print(*"Easy Sudoku Problem:"*)

easy.display(easy.infer\_assignment())

AC3(easy) #Calling Constraint Propagation to solve

print(*"\n"*)

print(*"After calling AC3:"*)

easy.display(easy.infer\_assignment())

if not easy.goal\_test(easy.curr\_domains):

print(*"\n"*)

print(*"Backtracking easy problem..."*)

result = backtracking\_search(easy,select\_unassigned\_variable=mrv)

if result:

print(*"After backtracking:"*)

easy.display(easy.infer\_assignment())

for puzzle in [harder1]:

hard = Sudoku(puzzle) # constructing the hard Sudoku problem

print(*"\n"*)

print(*"Hard Sudoku Problem:"*)

hard.display(hard.infer\_assignment())

AC3(hard) #Calling Constraint Propagation to solve

print(*"\n"*)

print(*"After calling AC3:"*)

hard.display(hard.infer\_assignment())

if not hard.goal\_test(hard.curr\_domains):

print(*"\n"*)

print(*"Backtracking hard problem..."*)

result = backtracking\_search(hard,select\_unassigned\_variable=mrv)

if result:

print(*"After backtracking:"*)

hard.display(hard.infer\_assignment())

Output:

|  |  |
| --- | --- |
| Easy Sudoku Problem:  . . 3 | . 2 . | 6 . .  9 . . | 3 . 5 | . . 1  . . 1 | 8 . 6 | 4 . .  ------+-------+------  . . 8 | 1 . 2 | 9 . .  7 . . | . . . | . . 8  . . 6 | 7 . 8 | 2 . .  ------+-------+------  . . 2 | 6 . 9 | 5 . .  8 . . | 2 . 3 | . . 9  . . 5 | . 1 . | 3 . .  After calling AC3:  4 8 3 | 9 2 1 | 6 5 7  9 6 7 | 3 4 5 | 8 2 1  2 5 1 | 8 7 6 | 4 9 3  ------+-------+------  5 4 8 | 1 3 2 | 9 7 6  7 2 9 | 5 6 4 | 1 3 8  1 3 6 | 7 9 8 | 2 4 5  ------+-------+------  3 7 2 | 6 8 9 | 5 1 4  8 1 4 | 2 5 3 | 7 6 9  6 9 5 | 4 1 7 | 3 8 2  Hard Sudoku Problem:  4 1 7 | 3 6 9 | 8 . 5  . 3 . | . . . | . . .  . . . | 7 . . | . . .  ------+-------+------  . 2 . | . . . | . 6 .  . . . | . 8 . | 4 . .  . . . | . 1 . | . . .  ------+-------+------  . . . | 6 . 3 | . 7 .  5 . . | 2 . . | . . .  1 . 4 | . . . | . . .  After calling AC3:  4 1 7 | 3 6 9 | 8 2 5  . 3 . | . . . | . . .  . . . | 7 . . | . . .  ------+-------+------  . 2 . | . . . | . 6 .  . . . | . 8 . | 4 . .  . . . | . 1 . | . . .  ------+-------+------  . . . | 6 . 3 | . 7 .  5 . . | 2 . . | . . .  1 . 4 | . . . | . . . | Backtracking hard problem...  After backtracking:  4 1 7 | 3 6 9 | 8 2 5  6 3 2 | 1 5 8 | 9 4 7  9 5 8 | 7 2 4 | 3 1 6  ------+-------+------  8 2 5 | 4 3 7 | 1 6 9  7 9 1 | 5 8 6 | 4 3 2  3 4 6 | 9 1 2 | 7 5 8  ------+-------+------  2 8 9 | 6 4 3 | 5 7 1  5 7 3 | 2 9 1 | 6 8 4  1 6 4 | 8 7 5 | 2 9 3 |