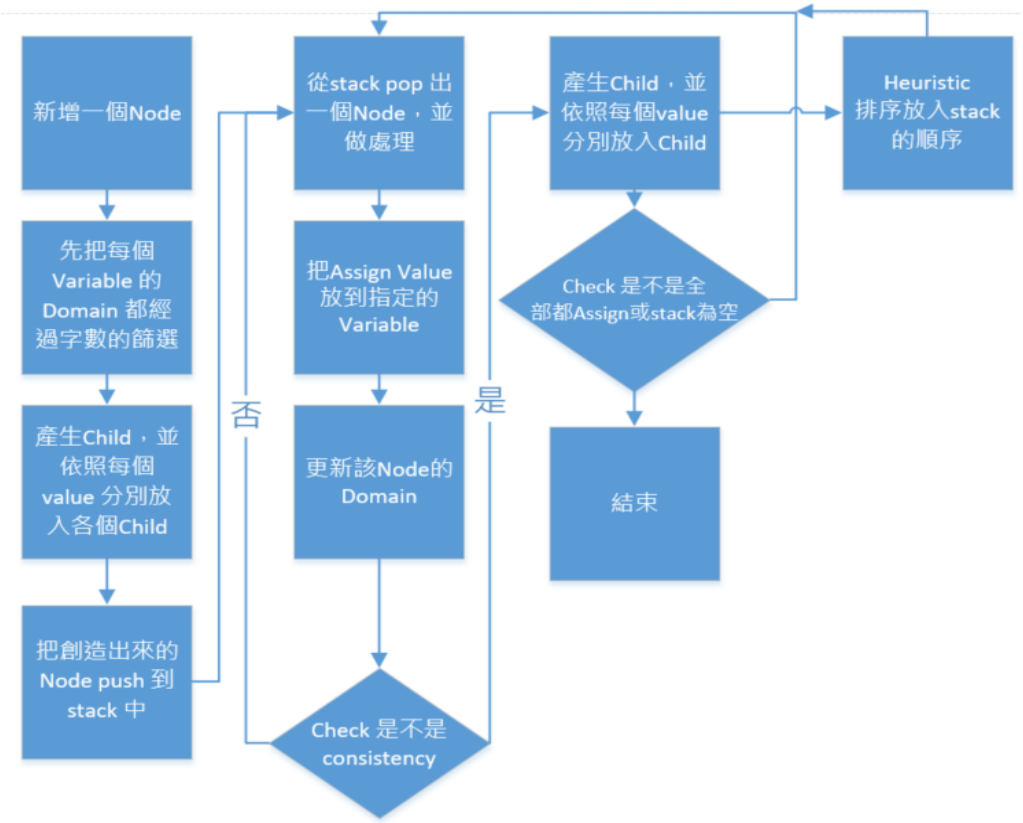
基本流程:

主要是利用 AC-3 的方法，如果有 Heuristic 則放入 Heuristc 流程: (每回測試有先把 words shuffle 過)

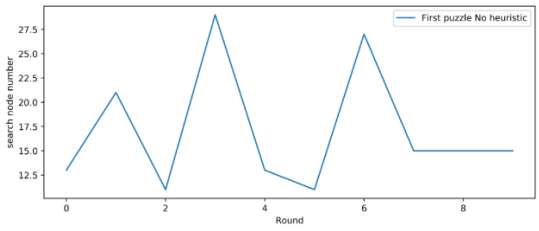
Heuristic 方法包含了 MRV、Degree、LCV

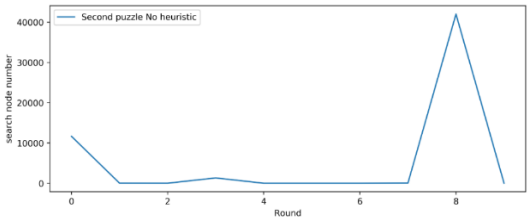


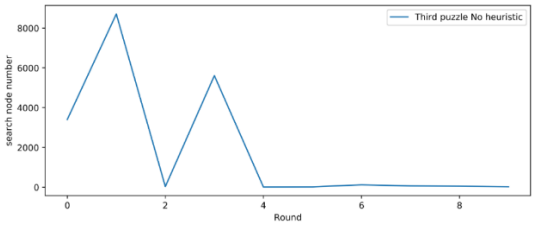
實驗

1. no heuristic (每回都有 shuffle 過且有 AC-3)

第一個 puzzles 、第二個 puzzles、第三個 puzzles:





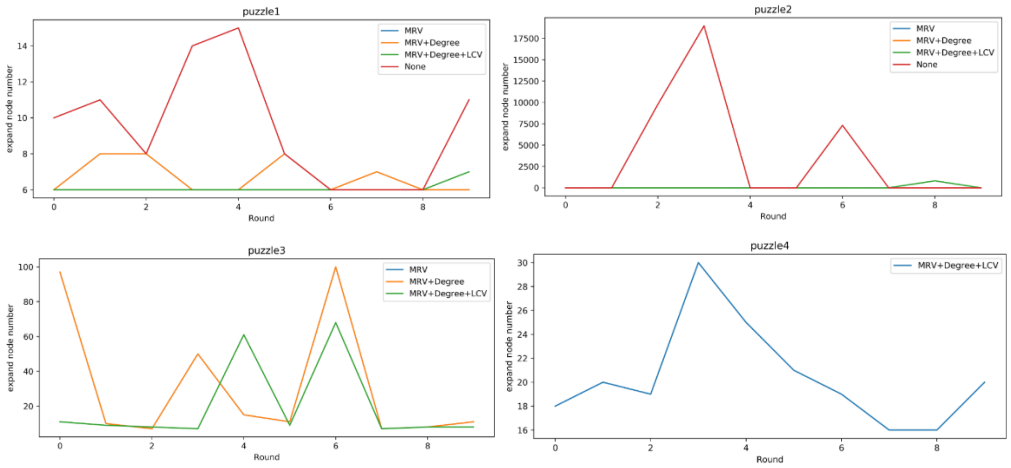


第四個 puzzles: > search more than 1 million node, some time will be fast.

討論:
因為這個搜尋的方法類似 DFS，不斷的往下找，雖然有一邊刪除每個 Variable 的 Domain，但會遇到一個主要的問題是，當我一開始選擇到一個最後不會找答案的 value 當作 Variable 時，我還是要繼續往前把後面的 domain 都走過一次，這就很像老鼠走迷宮一樣，一開始如果往左邊走永遠不會到達終點，但我還是要先把左邊的都先走完，才能走一開始的右邊搜尋。另外在 puzzle4 中，搜尋時間很久，雖然越多 node，有 constraint 的可能也會越大，但越多 variable 可能的答案越少，因此如果單純只用 None heuristic 的方法找，似乎要花很久的時間才能找到答案。

2. heuristic (MRV → Degree → LCV)

第一個 puzzles 、第二個 puzzles、第三個 puzzles:



討論:
(1)在這個測試當每次產生 Child Node 時 利用 MRV，選出 domain 剩下較少的優先放在 stack 上方，因此下一輪的 expand 會先擴展 domain 剩下較少的。

(2)MRV+Degree 是當 domain 剩下同樣數量時，接著會比較 Degree，方法是比較兩者 constrain 的 not assigned variable 數量，constrain 越多會優先放在 stack 上方，從上方這些圖看起來，MRV 和 MRV + Degree 的方法 Search Node 數量幾乎一樣，這是因為當 domain 剩下的數量相同時，可能的情況只會有自己跟自己，不太可能會剛好有相同的 domain 數量的 variable，因此這時就要靠 LCV 來決定該同樣 domain 下，但不同 value 的順序。

(3)MRV + Degree + LCV，LCV 的方法則是利用預先 forward 測試一次，如果能把全部的 domain 數量降到最少，則能優先放在 stack 上方。

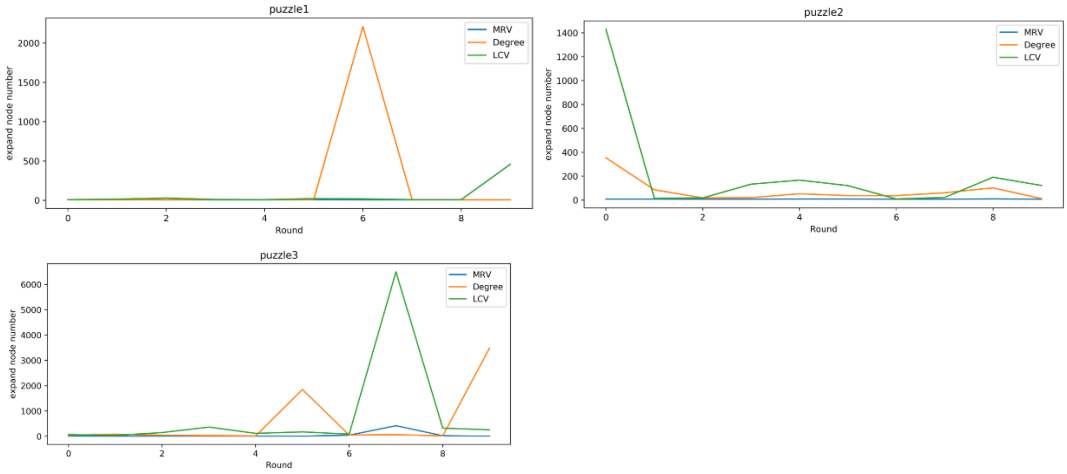
(4)puzzle1 圖，可以看出只用 AC-3 的方法比起有加其他 heuristic 方法還要差，不過有時會比較好，應該是因為運氣好而選到的。另外 MRV 幾乎等於 MRV + Degree，在 (2) 有提到。整體看來是 None < MRV = MRV + Degree < MRV+Degree + LCV

(5) puzzle3 圖，有時 MRV+Degree + LCV > MRV 的情況，應該是因為經過 LCV forward check 時 domain 剩下的數量太過於極端，例如某個 variable 的 domain 已經剩下 2，但另一個 variable 的 domain 剩下 200，加起來仍有 202，因此在 LCV 時可能會做出錯的決定。

(6)puzzle4 圖，因其他優化方法，有時數值會變得很大，我覺得應該是跟 DFS 的性質有關係，因為答案的數量少，只要一開始走錯就很難找到答案，或要花很久的時間，因此需要利用 MRV + Degree + LCV 的優化方法。

3. heuristic (MRV 、 Degree 、 LCV 個別執行)

第一個 puzzles 、第二個 puzzles、第三個 puzzles:



討論:
由這幾張圖可以發現如果 MRV、Degree、LCV 分開做，單獨使用 MRV 的效果會比單獨使用 Degree 與 LCV 還要好，因此解 puzzle 的題目還是要從 domain 少的為第一條件，然後才是 constrain 的數量，最後才是 LCV。

4. 單一答案(只舉一個例子):

Puzzle 1: left 、 light 、 from 、 on 、 him
Puzzle 2: truly 、 tribe 、 illegal 、 begin 、 late 、 not

Puzzle 3: odds 、 owner 、 style 、 who 、 hungry 、 rare
Puzzle 4: sales 、 satisfy 、 chef 、be 、 schedule 、 fuel 、 increase 、 formula 、 type 、 carry 、 at 、 to

5. Puzzle 4 search 其中一次過程

(空白為沒有 assigned value)
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'to']
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'to']
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'at', 'to']
[' ', ' ', ' ', ' ', ' ', ' ', ' ', 'formula', ' ', ' ', 'at', 'to']
[' ', 'testify', ' ', ' ', ' ', ' ', ' ', 'formula', ' ', ' ', 'at', 'to']
[' ', 'testify', ' ', ' ', 'surround', ' ', ' ', 'formula', ' ', ' ', 'at', 'to']
[' ', 'testify', ' ', ' ', 'surround', ' ', 'tomorrow', 'formula', ' ', ' ', 'at', 'to']
['terms', 'testify', ' ', ' ', 'surround', ' ', 'tomorrow', 'formula', ' ', ' ', 'at', 'to']
['terms', 'testify', ' ', ' ', 'surround', ' ', 'tomorrow', 'formula', ' ', 'moral', 'at', 'to']
['terms', 'testify', ' ', ' ', 'surround', ' ', 'tomorrow', 'formula', 'glad', 'moral', 'at', 'to']
['terms', 'testify', ' ', ' ', 'surround', 'town', 'tomorrow', 'formula', 'glad', 'moral', 'at', 'to']
['terms', 'testify', 'unit', ' ', 'surround', 'town', 'tomorrow', 'formula', 'glad', 'moral', 'at', 'to']
['terms', 'testify', 'unit', 'hi', 'surround', 'town', 'tomorrow', 'formula', 'glad', 'moral', 'at', 'to']

討論:
總共 18 次 search，其中會重複是因為發生 consistency，因此在 search tree 中要跳回上一層。

附錄:

```
import pandas as pd
import numpy  as np
import copy
import functools
import random
import csv

class puzzleInfo():
    def __init__(self,puzzleIndex,wordGroup):
        if puzzleIndex == 0:
            self.constraintMap = [[-1,0,0,-1,-1],[0,-1,-1,-1,0],[2,-1,-1,0,2],[-1,-1,2,-1,-1],[-1,3,3,-1,-1]]
            self.variableLen   = [4,5,4,2,3]
        elif puzzleIndex == 1:
            self.constraintMap = [[-1,1,2,-1,-1,-1],[1,-1,-1,0,-1,-1],[3,-1,-1,2,0,-1],[-1,3,4,-1,-1,0],[-1,-1,6,-1,-1,2],[-1,-1,-1,4,2,-1]]
            self.variableLen   = [5,5,7,5,4,3]
        elif puzzleIndex == 2:
            self.constraintMap = [[-1,0,0,-1,-1,-1],[0,-1,-1,-1,2,0],[3,-1,-1,-1,5,3],[-1,-1,-1,-1,0,-1],[-1,2,2,1,-1,-1],[-1,4,4,-1,-1,-1]]
            self.variableLen   = [4,5,5,3,6,4]
        else:
            self.constraintMap = [[-1,0 ,-1,-1,0 ,-1,-1,-1,-1,-1,-1,-1],
                                   [0 ,-1,-1,-1,-1,-1,-1,0 ,0 ,-1,-1,-1,-1],
                                   [-1,-1,-1,1 ,1 ,0 ,-1,-1,-1,-1,-1,-1],
                                   [-1,-1,2 ,-1,-1,-1,-1,-1,-1,-1,-1,-1],
                                   [4 ,-1,0 ,-1,-1,-1,4 ,4 ,3 ,-1,-1,-1],
                                   [-1,-1,3 ,-1,-1,-1,7 ,-1,-1,-1,-1,-1],
                                   [-1,3 ,-1,-1,3 ,2 ,-1,-1,-1,0 ,-1,-1],
                                   [-1,5 ,-1,-1,5 ,-1,-1,-1,-1,2 ,0 ,-1],
                                   [-1,-1,-1,-1,7 ,-1,-1,-1,-1,4 ,-1,-1],
                                   [-1,-1,-1,-1,-1,-1,-1,2 ,2 ,1 ,-1,-1,-1],
                                   [-1,-1,-1,-1,-1,-1,-1,-1,6 ,-1,-1,-1,0 ],
                                   [-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1 ,-1]]
            self.variableLen   = [5,7,4,2,8,4,8,7,4,5,2,2]


class puzzleNode():
    def __init__(self):
        self.assignVarIndex =  None
        self.assignWrdIndex =  None
        self.domain         =  []

    def init_domain(self,var_len,wordGroup):
        self.alreadyAssign  =  np.empty(len(var_len))
        self.alreadyAssign.fill(-1)
        print("First domain example:")
        for length in var_len:
            self.domain.append(wordGroup[length].values)
            print(length,wordDataFrame.at[wordGroup[length].values[0],'words'])

        print("First node domain len:",len(self.domain))

    def print_domain(self):
        print(self.domain)

    def set_domain(self):
        # set assignVarIndex = 0 , assignWrdIndex = 500
        if self.assignVarIndex is not None :
            self.domain[self.assignVarIndex] = np.array([self.assignWrdIndex])

```

```
    def update_domain(self,constraintMap):
        varIndex = self.assignVarIndex
        wrdIndex = self.assignWrdIndex
        self.alreadyAssign[varIndex] = wrdIndex
        assignWord = wordDataFrame.at[wrdIndex,'words']
        for index,element in enumerate(self.alreadyAssign):
            if element == -1 :
                prohibitIndex = constraintMap[varIndex][index]
                if prohibitIndex == -1:
                    continue
                doProhitbitWord = assignWord[constraintMap[index][varIndex]]
                delete_list = []
                for index2,element2 in enumerate(self.domain[index]):
                    if wordDataFrame.at[element2,'words'][prohibitIndex] != doProhitbitWord:
                        delete_list.append(index2)

                self.domain[index] = np.delete(self.domain[index],delete_list)
            else:
                continue

    def checkConsistency(self):
        for element in self.domain:
            if element.size == 0:
                return False
        solution = 1
        for index,element in enumerate(self.alreadyAssign):
            if element == -1:
                solution = 0

        if solution == 1:
            #print("Following is solution dictionary index:")
            ans = []
            for e in self.alreadyAssign:
                ans.append(wordDataFrame.at[int(e),'words'])
            check = 0
            for element in answer:
                check = 1
                for index2,element2 in enumerate(element):
                    if element2 != ans[index2]:
                        check = 0
                if check == 1:
                    break
            if check != 1:
                answer.append(ans)
            return False
        else:
            return True
```

```
def compareDomain(Node1, Node2):
    firstLen  = len(Node1.domain[Node1.assignVarIndex])
    secondLen = len(Node2.domain[Node2.assignVarIndex])
    if firstLen < secondLen :
        return 1
    elif firstLen > secondLen:
        return -1
    elif firstLen == secondLen  and mode > 0:
        firstconstrainNum  = 0
        secondconstrainNum = 0
        for index,element in enumerate(Node1.alreadyAssign):
            if element == -1 and puzzleBoard.constraintMap[Node1.assignVarIndex][index] != -1 :
                firstconstrainNum = firstconstrainNum + 1
        for index2,element2 in enumerate(Node2.alreadyAssign):
            if element2 == -1 and puzzleBoard.constraintMap[Node2.assignVarIndex][index2] != -1 :
                secondconstrainNum = secondconstrainNum + 1

        if firstconstrainNum < secondconstrainNum:
            return -1
        elif firstconstrainNum > secondconstrainNum:
            return 1
        elif mode > 1:
            domain1 = copy.copy(Node1.domain)
            domain2 = copy.copy(Node2.domain)
            count1 = 0
            count2 = 0
            varIndex = Node1.assignVarIndex
            wrdIndex = Node1.assignWrdIndex
            assignWord = wordDataFrame.at[wrdIndex,'words']

            varIndex2 = Node2.assignVarIndex
            wrdIndex2 = Node2.assignWrdIndex
            assignWord2 = wordDataFrame.at[wrdIndex2,'words']

            delete_list = []
            for index,element in enumerate(Node1.alreadyAssign):
                if element == -1 and index != Node1.assignVarIndex:
                    prohibitIndex = puzzleBoard.constraintMap[varIndex][index]
                    if prohibitIndex == -1:
                        continue
                    doProhitbitWord = assignWord[puzzleBoard.constraintMap[index][varIndex]]
                    for index2,element2 in enumerate(domain1[index]):
                        if wordDataFrame.at[element2,'words'][prohibitIndex] != doProhitbitWord:
                            delete_list.append(index2)
                    domain1[index] = np.delete(domain1[index],delete_list)
                count1 = count1 + len(domain1[index])
            delete_list = []
            for index,element in enumerate(Node2.alreadyAssign):
                if element == -1 and index != Node2.assignVarIndex:
                    prohibitIndex = puzzleBoard.constraintMap[varIndex2][index]
                    if prohibitIndex == -1:
                        continue
                    doProhitbitWord = assignWord2[puzzleBoard.constraintMap[index][varIndex2]]
                    for index2,element2 in enumerate(domain2[index]):
                        if wordDataFrame.at[element2,'words'][prohibitIndex] != doProhitbitWord:
                            delete_list.append(index2)
                    domain2[index] = np.delete(domain2[index],delete_list)
                count2 = count2 + len(domain2[index])
```

```
            if count1 < count2:
                return -1
            elif count1 > count2:
                return 1
    return 0

def genChild(parent):
    childNode = []
    for index,element in enumerate(parent.alreadyAssign):
        if element == -1:
            for index2,element2 in enumerate(parent.domain[index]):
                newNode = puzzleNode()
                newNode.assignVarIndex = index
                newNode.assignWrdIndex = element2
                newNode.domain = copy.copy(parent.domain)
                newNode.alreadyAssign = copy.copy(parent.alreadyAssign)
                childNode.append(newNode)
    #random.shuffle(childNode)
    if mode != 3 :
        childNode = sorted(childNode,key=functools.cmp_to_key(compareDomain))
        #print(childNode[-1].assignVarIndex)
    return childNode

def genWordGroup(wordDataFrame):
    wordDataFrame['wordLength'] = 0
    for index, row in wordDataFrame.iterrows():
        wordDataFrame.at[index,'wordLength'] = len(row['words'])

    re = wordDataFrame.groupby('wordLength').groups
    return re

# starTest is for testing many kind of mode with heuristic
def starTest(wordGroup):
    expandNodeNum = 0
    puzzleStack = []
    if mode == 0:
        print("Heuristic: MRV")
        firstNode     = puzzleNode()
        firstNode.init_domain(puzzleBoard.variableLen,wordGroup)
        puzzleStack.append(firstNode)
        print("Start to expand nodes")
        i = 0
        while puzzleStack:
            expandNode = puzzleStack[-1]
            puzzleStack.pop()
            expandNodeNum = expandNodeNum + 1
            if expandNode.assignVarIndex is not None:
                expandNode.set_domain()
                expandNode.update_domain(puzzleBoard.constraintMap)


            if expandNode.checkConsistency() == True:
                child = genChild(expandNode)
            else:
                if len(answer) == anserLimit:
                    print("answer equal the limit: ",anserLimit)
```

```
                    break
                continue
            for index,element in enumerate(child):
                puzzleStack.append(element)
        print("End of the expansion")
        print("Number of expand node: ",expandNodeNum)
        print("-----------------------")
    elif mode == 1:
        print("Heuristic: MRV、Degree")
        firstNode     = puzzleNode()
        firstNode.init_domain(puzzleBoard.variableLen,wordGroup)
        puzzleStack.append(firstNode)
        print("Start to expand nodes")
        i = 0
        while puzzleStack:
            expandNode = puzzleStack[-1]
            puzzleStack.pop()
            expandNodeNum = expandNodeNum + 1
            if expandNode.assignVarIndex is not None:
                expandNode.set_domain()
                expandNode.update_domain(puzzleBoard.constraintMap)


            if expandNode.checkConsistency() == True:
                #print(456)
                child = genChild(expandNode)
            else:
                #print(123)
                if len(answer) == anserLimit:
                    break
                continue
            for index,element in enumerate(child):
                puzzleStack.append(element)
        print("End of the expansion")
        print("Number of expand node: ",expandNodeNum)
        print("-----------------------")
    elif mode == 2:
        print("Heuristic: MRV、Degree、LCV")
        firstNode     = puzzleNode()
        firstNode.init_domain(puzzleBoard.variableLen,wordGroup)
        puzzleStack.append(firstNode)
        print("Start to expand nodes")
        i = 0
        while puzzleStack:
            expandNode = puzzleStack[-1]
            puzzleStack.pop()
            expandNodeNum = expandNodeNum + 1
            if expandNode.assignVarIndex is not None:
                expandNode.set_domain()
                expandNode.update_domain(puzzleBoard.constraintMap)

            if expandNode.checkConsistency() == True:
                child = genChild(expandNode)
            else:
                if len(answer) == anserLimit:
                    break
                continue
```

```
            for index,element in enumerate(child):
                puzzleStack.append(element)
        print("End of the expansion")
        print("Number of expand node: ",expandNodeNum)
        print("------------------------")
    else:
        print("Heuristic: ")
        firstNode     = puzzleNode()
        firstNode.init_domain(puzzleBoard.variableLen,wordGroup)
        puzzleStack.append(firstNode)
        print("Start to expand nodes")
        i = 0
        while puzzleStack:
            expandNode = puzzleStack[-1]
            puzzleStack.pop()
            expandNodeNum = expandNodeNum + 1
            if expandNodeNum % 10000 == 0:
                print(expandNodeNum)
            if expandNode.assignVarIndex is not None:
                expandNode.set_domain()
                expandNode.update_domain(puzzleBoard.constraintMap)

            if expandNode.checkConsistency() == True:
                child = genChild(expandNode)
            else:
                if len(answer) == anserLimit:
                    break
                continue

            for index,element in enumerate(child):
                puzzleStack.append(element)
        print("End of the expansion")
        print("Number of expand node: ",expandNodeNum)
        print("------------------------")
    return expandNodeNum


def main():
    global mode
    global puzzleBoard
    global anserLimit
    global answer
    global wordDataFrame
    # init constraints map
    # setup mode、 answer 、 answerLimit 、 puzzleBoard

    testingSet = []
    for step in range(0,10):
        answer = []
        anserLimit = 1
        wordDataFrame = pd.read_csv('./English-words-3000.txt',sep = "\n",names = ['words'])
        wordDataFrame = wordDataFrame.sample(frac=1).reset_index(drop=True)
        wordGroup     = genWordGroup(wordDataFrame)
        puzzleBoard = puzzleInfo(0,wordGroup)
```

```
        tmpList = []
        for i in range(0,3):
            answer = []
            mode   = i
            print("mode: ",mode)
            path   = './ans' + str(i) + '.csv'
            expandNodeNum = starTest(wordGroup)
            tmpList.append(expandNodeNum)
            df = pd.DataFrame(answer,columns=['0','1','2','3','4'])
            df.to_csv(path,sep = ',')
        testingSet.append(tmpList)
    df = pd.DataFrame(testingSet,columns=['0','1','2','3','4'])
    df.to_csv('output.csv',sep=',')
    print(testingSet)
if __name__ == '__main__':
    main()
```