人工智能：$pj_2 - nQueens$

李云帆 16302010002

前言

本次作业主要是针对CSP问题的求解，感谢高助教的code skeleton，真的很容易理解，大大加快了我理解用搜索解CSP问题的速度。

建模

这个问题的建模很简单，就是一个n*n的棋盘，放上n个皇后，使得每个皇后不能（按照国际象棋的规则）吃掉其他皇后。

我们画一个图就知道：

	Q		
			Q
Q			
		Q	

这个是四皇后的一个解

对于水平方向，我们令它为X轴，竖直方向为Y轴，建模棋盘为二维数组

并且我们假定

代码实现

```
def create_n_queens_csp(n=8):
    """Create an N-Queen problem on the board of size n * n.

    You should call csp.add_variable() and csp.add_binary_factor().

    Args:
        n: int, number of queens, or the size of one dimension of the board.

    Returns
        csp: A CSP problem with correctly configured factor tables
        such that it can be solved by a weighted CSP solver
    """
    csp = CSP()
    # TODO: Problem a
    # TODO: BEGIN_YOUR_CODE
    vars = []
    for i in range(1, n + 1):
        # we suppose that on a board, we put in queens with order.
        # In other words, Xi is fixed for each queen,
        # we only have to consider Yi for each queen
        varName = 'Y' + str(i)
        csp.add_variable(varName, range(1, n + 1))
        vars.append(varName)

    rule1 = lambda x, y: x != y
    for var in vars:
        for anotherVar in vars:
            if var != anotherVar:
                # csp.add_binary_factor(var, anotherVar, lambda y1, y2: y1 != y2)
                dist = abs(vars.index(anotherVar) - vars.index(var))
                csp.add_binary_factor(var, anotherVar, lambda y1, y2: y1 != y2 and abs(y1 - y2) != dist)

    # raise NotImplementedError
    # TODO: END_YOUR_CODE
    return csp
```

这段代码主要生成了我们棋盘上的n个皇后，并且生成了皇后之间的二元约束关系

```
def get_unassigned_variable(self, assignment):
    """Get a currently unassigned variable for a partial assignment.

    If mcv is True, Use heuristic: most constrained variable (MCV)
    Otherwise, select a variable without any heuristics.

    Most Constrained Variable (MCV):
        Select a variable with the least number of remaining domain values.
        Hint: self.domains[var] gives you all the possible values
        Hint: get_delta_weight gives the change in weights given a partial
```

```
            assignment, a variable, and a proposed value to this variable
        Hint: choose the variable with lowest index in self.csp.variables
            for ties

        Args:
            assignment: a dictionary of current assignment.

        Returns
            var: a currently unassigned variable.
        """
        if not self.mcv:
            for var in self.csp.variables:
                if var not in assignment:
                    return var
        else:
            # TODO: Problem b
            # TODO: BEGIN_YOUR_CODE
            mcv1 = self.csp.variables[0]
            domain_mcv = float('inf')  # the basic idea is to find min domain
            for var in self.csp.variables:
                if var not in assignment:
                    var_domain = 0
                    for value in self.domains[var]:
                        if self.check_factors(assignment, var, value) != 0:
                            var_domain += 1
                    if var_domain < domain_mcv:  # choose a var with the least domain space
                        domain_mcv = var_domain
                        mcv1 = var
            return mcv1
            # raise NotImplementedError
            # TODO: END_YOUR_CODE
```

这段代码主要做了选出当前状态下的Most Constrained Variable (MCV).

利用self.csp.variables获取全部变量，通过check_factors()检查两个变量之间的约束关系是否满足

```
def arc_consistency_check(self, var):
    """AC-3 algorithm.

    The goal is to reduce the size of the domain values for the unassigned
    variables based on arc consistency.

    Hint: get variables neighboring variable var:
        self.csp.get_neighbor_vars(var)

    Hint: check if a value or two values are inconsistent:
        For unary factors
            self.csp.unaryFactors[var1][val1] == 0
        For binary factors
            self.csp.binaryFactors[var1][var2][val1][val2] == 0

    Args:
        var: the variable whose value has just been set
    """
    # TODO: Problem c
    # TODO: BEGIN_YOUR_CODE
    queue = [(neighbor, var) for neighbor in self.csp.get_neighbor_vars(var)]

    # q = Queue.Queue()
    # for arc in self.csp.binary_factors:
    #     q.put(arc)

    def remove_inconsistent_values(x1, x2):
        removed = False
        """
        tmp1 = self.domains[x1]
        tmp2 = [value for value in self.domains[x1]]
        if sum(tmp1) != sum(tmp2):
            print("stop")
        for x1v in self.domains[x1]:
        """
        unchangedDomains = list(self.domains[x1])

        # for x1v in [value for value in self.domains[x1]]:
        # this is a unchanged version of self.domains[x1]
        # as we change self.domains[x1] in the following for loop,
        # we would like our domain to stay the same in the conditions of the for loop
        for x1v in unchangedDomains:
            consistent = False
            for x2v in self.domains[x2]:
                if self.csp.binary_factors[x1][x2][x1v][x2v] != 0:
                    consistent = True
                    # break
            if not consistent:
```

```
                self.domains[x1].remove(x1v)
                removed = True
        return removed

    while len(queue):
        x1, x2 = queue.pop(0)  # pop out the first arc (list type can also do pop!!!)
        if remove_inconsistent_values(x1, x2):
            for xk in self.csp.get_neighbor_vars(x1):
                if xk != x2:
                    queue.append((xk, x1))
```

这段代码实现了书上的AC-3算法，其中有一个非常有意思的地方是：我们在循环的内部修改了

```
def arc_consistency_check(self, var):
    """AC-3 algorithm.

    The goal is to reduce the size of the domain values for the unassigned
    variables based on arc consistency.

    Hint: get variables neighboring variable var:
        self.csp.get_neighbor_vars(var)

    Hint: check if a value or two values are inconsistent:
        For unary factors
            self.csp.unaryFactors[var1][val1] == 0
        For binary factors
            self.csp.binaryFactors[var1][var2][val1][val2] == 0

    Args:
        var: the variable whose value has just been set
    """
    # TODO: Problem c
    # TODO: BEGIN_YOUR_CODE
    queue = [(neighbor, var) for neighbor in self.csp.get_neighbor_vars(var)]

    # q = Queue.Queue()
    # for arc in self.csp.binary_factors:
    #     q.put(arc)

    def remove_inconsistent_values(x1, x2):
        removed = False
        """
        tmp1 = self.domains[x1]
        tmp2 = [value for value in self.domains[x1]]
        if sum(tmp1) != sum(tmp2):
            print("stop")
        for x1v in self.domains[x1]:
        """
        unchangedDomains = list(self.domains[x1])

        # for x1v in [value for value in self.domains[x1]]:
        # this is a unchanged version of self.domains[x1]
        # as we change self.domains[x1] in the following for loop,
        # we would like our domain to stay the same in the conditions of the for loop
        for x1v in unchangedDomains:
            consistent = False
            for x2v in self.domains[x2]:
                if self.csp.binary_factors[x1][x2][x1v][x2v] != 0:
                    consistent = True
                    # break
            if not consistent:
                self.domains[x1].remove(x1v)
                removed = True
        return removed

    while len(queue):
        x1, x2 = queue.pop(0)  # pop out the first arc (list type can also do pop!!!)
        if remove_inconsistent_values(x1, x2):
            for xk in self.csp.get_neighbor_vars(x1):
                if xk != x2:
                    queue.append((xk, x1))
```

这段代码实现了书上的AC-3算法，其中有一个非常有意思的地方是：我们在循环的内部修改了self.domains[x1]，而我们事实上希望的是我们的循环不受到影响，这里有两种解决方法，第一个我们使用iterator，第二种是我们生成一个新的self.domains[x1]的副本。

我采取的是后者。

至此，这个n-queens的pj就写完了，我们可以通过运行test.py来查看我们的运行结果。

```
➜  pj2-n-queens git:(master) ✗ python test.py
========== START GRADING
----- START PART a: Test for Create 8-Queens CSP
----- END PART a [took 0:00:00.032399 (max allowed 1 seconds), 5/5 points]
()
----- START PART b: Test for MCV with 8-Queens CSP
----- END PART b [took 0:00:00.061114 (max allowed 1 seconds), 5/5 points]
()
----- START PART c: Test for AC-3 with n-queens CSP
----- END PART c [took 0:00:00.094022 (max allowed 1 seconds), 10/10 points]
()
========== END GRADING [20/20 points]
```