COMP 9414

Assignment 1 讲解

主讲人：Tara

2020.6

一 Assignment 要求

Assignment 描述

一个工厂要完成一系列任务的调度，有一些任务之间存在约束，比如有截止日期，任务的先后次序等等。有的约束是必须满足的，有的约束违背是可以接受的，但是会产生cost。在这个场景中，每个任务的持续时间是给定的，每个任务必须在同一天的工作时间（9am--5pm）开始和结束。

任务的约束有三种：

1. 针对单个任务的约束（完成时间的限制）
2. 针对两个问题的约束（先后次序）
3. 非强制实现的约束(soft constraint)：如果违背，会产生cost

我们的目标：为所有的任务在一周内制定时间表，在满足所有强制性的约束条件下，最小化cost

— Assignment 思路

知识点介绍：

定义一个约束满足问题

一组变量，每个变量都有自己的值。当每个变量的赋值都满足所有关于变量的约束时，问题就得到了解决。这类问题称为**约束满足问题**（CSP）

约束满足问题包含三个成分：

变量 Variable：{X1, X2....., Xn}

值域 Domains: {D1, D2....., Dn }, 每个变量都有自己的值域

约束 Constraints: 描述变量取值的约束集合。

每个约束Ci为<scope, rel>，scope为约束中的变量组，rel表示变量满足的关系。

例子：X1, X2 的值域都是{A, B}，二者不能相等，则约束为

<(X1, X2), [(A, B), (B, A)]>或<(X1, X2), X1≠X2>

知识点介绍：

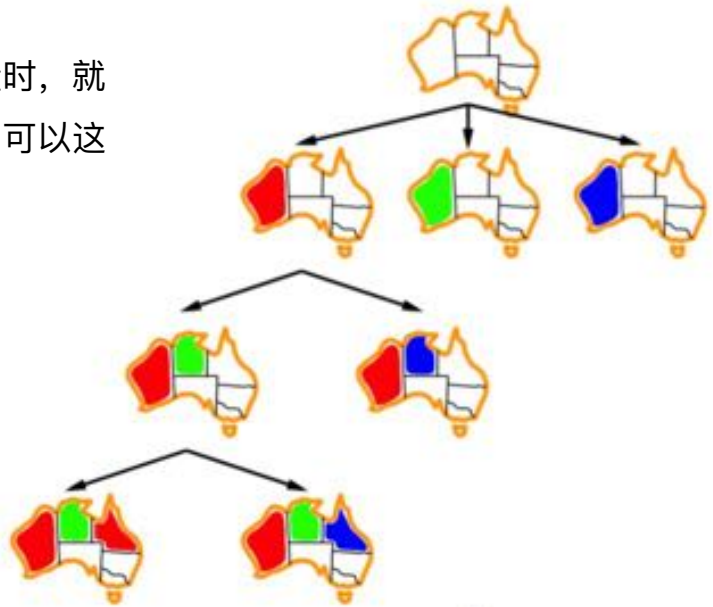
CSP的回溯搜索(Backtracking Search)

每次选择一个变量变量进行赋值，当没有合法的值可以赋给某变量时，就回溯。当所有的变量都被赋值时，找到解。与深度优先算法类似，可以这样定义：

initial state: {}

Successor function: 给某变量赋值后，跟其它已被赋值的变量的constraint不冲突。

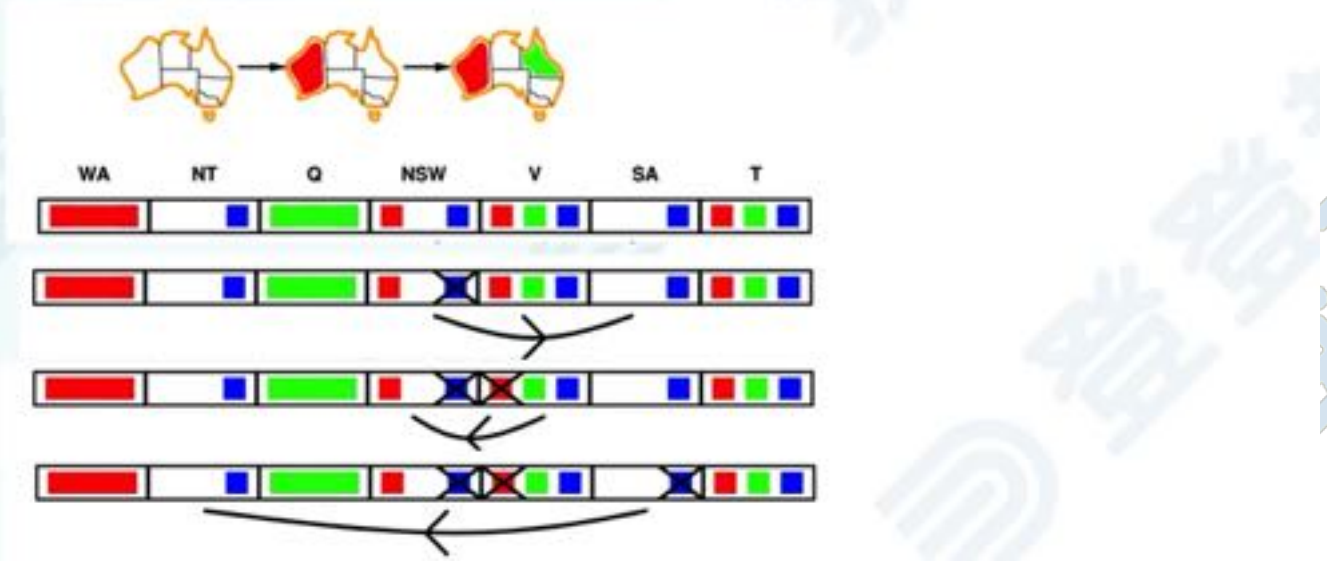
Goal state: 所有的目标都被赋值，所有的contraint都满足



保证constraint：

弧相容(Arc Consistency)

当变量Xi赋值时，从Xi邻接的弧中所有未赋值变量Xj开始，进行约束传播，一旦某个变量的值域变空，则赋值失败立即回溯。



思路

1. 将场景转变成CSP问题
2. 对该csp问题进行搜索求解

   搜索过程中使用Arc Consistency

   搜索最优解(cost 最小)

1 将场景转变成CSP问题

```
# two tasks with two binary constraints and soft deadlines
task, t1 3
task, t2 4
# two binary constraints
constraint, t1 before t2
constraint, t1 same-day t2
# domain constraint
domain, t2 mon
# soft deadlines
domain, t1 ends-by mon 3pm 10
domain, t2 ends-by mon 3pm 10
```

import sys
file_path = sys.argv[1]
从input中，可以获得：
variable
domain: (start_time, end_time)
hard_constraint(binary, unary),
soft_constraint
soft_cost

代码实现

aipython cspProblem.py
cspExample.py

```
csp2 = CSP({'A':{1,2,3,4},'B':{1,2,3,4}, 'C':{1,2,3,4},
            'D':{1,2,3,4}, 'E':{1,2,3,4}},
           [Constraint(('B',),ne_(3)),
            Constraint(('C',),ne_(2)),
            Constraint(('A','B'),ne),
            Constraint(('B','C'),ne),
            Constraint(('C','D'),lt),
            Constraint(('A','D'),eq),
            Constraint(('A','E'),gt),
            Constraint(('B','E'),gt),
            Constraint(('C','E'),gt),
            Constraint(('D','E'),gt),
            Constraint(('B','D'),ne)])
```

```
class CSP(object):
    """A CSP consists of
    * domains, a dictionary that maps each variable to its domain
    * constraints, a list of constraints
    * variables, a set of variables
    * var_to_const, a variable to set of constraints dictionary
    """
    def __init__(self,domains,constraints):
        """domains is a variable:domain dictionary
        constraints is a list of constriants
        """
        self.variables = set(domains)
        self.domains = domains
        self.constraints = constraints
        self.var_to_const = {var:set() for var in self.variables}
        for con in constraints:
            for var in con.scope:
                self.var_to_const[var].add(con)

    def __str__(self):
        """string representation of CSP"""
        return str(self.domains)

    def __repr__(self):
        """more detailed string representation of CSP"""
        return "CSP("+str(self.domains)+", "+str([str(c) for c in self.constraints])+")"

    def consistent(self,assignment):
        """assignment is a variable:value dictionary
        returns True if all of the constraints that can be evaluated
                        evaluate to True given assignment.
        """
        return all(con.holds(assignment)
                   for con in self.constraints
                   if all(v in assignment for v in con.scope))
```

```
csp2 = CSP({'A':{1,2,3,4}, 'B':{1,2,3,4}, 'C':{1,2,3,4},
            'D':{1,2,3,4}, 'E':{1,2,3,4}},
           [Constraint(('B',),ne_(3)),
            Constraint(('C',),ne_(2)),
            Constraint(('A','B'),ne),
            Constraint(('B','C'),ne),
            Constraint(('C','D'),lt),
            Constraint(('A','D'),eq),
            Constraint(('A','E'),gt),
            Constraint(('B','E'),gt),
            Constraint(('C','E'),gt),
            Constraint(('D','E'),gt),
            Constraint(('B','D'),ne)])
```

```
class Constraint(object):
    """A Constraint consists of
    * scope: a tuple of variables
    * condition: a function that can applied to a tuple of values
    for the variables
    """
    def __init__(self, scope, condition):
        self.scope = scope
        self.condition = condition
```

```
def ne_(val):
    """not equal value"""
    # nev = lambda x: x != val    # alternative definition
    # nev = partial(neq,val)      # another alternative definition
    def nev(x):
        return val != x
    nev.__name__ = str(val)+"!="      # name of the function
    return nev
```

如果f = ne (3),那么 f (2) 是True f(3) 是False.
当x不等于y，ne (x)(y) 为 true

```
# two tasks with two binary constraints and soft deadlines
task, t1 3
task, t2 4
# two binary constraints
constraint, t1 before t2
constraint, t1 same-day t2
# domain constraint
domain, t2 mon
# soft deadlines
domain, t1 ends-by mon 3pm 10
domain, t2 ends-by mon 3pm 10
```

我们需要做的是：

1. extend CSP class，加入soft_constraint 和 soft_cost
2. 从文件中依次读入需要加入new CSP的信息
   domain: {t1:{(start_time, end_time), ...}}
   hard_domain: {Constraint((scope), condition)}
   soft_domain: {t1: time, t2: time }
   soft_cost: {t1: cost, t2: cost}
3. 构建完成
   csp = New_CSP(domain,hard_constraint,soft_constraint,soft_cost)

1.1 extend CSP class

```
class New_CSP(CSP):
    def __init__(self,domains,constraints,soft_constraints,soft_cost):
        super().__init__(domains,constraints)
        self.soft_constraints = soft_constraints
        self.soft_cost = soft_cost
```

1.2.1 Get domain from input

domain: {t1:{(start_time, end_time), ...}}

```
week_to_num = {'mon': 1, 'tue': 2, 'wed': 3, 'thu': 4, 'fri': 5}
time_to_num = {'9am': 1, '10am': 2, '11am': 3, '12pm': 4, '1pm': 5, '2pm
```

{'t1': [(11, 14), (12, 15), (13, 16), (14, 17), (15, 18), (16, 19), (21, 24), (22, 25), (23, 26), (24, 27), (25, 28), ...

注意 ： 每个task要在当天完成，所以domain这时候有个限制，即end_time里的时间不能晚于5pm。

```
# two tasks with two binary constraints and soft deadlines
task, t1 3
task, t2 4
# two binary constraints
constraint, t1 before t2
constraint, t1 same-day t2
# domain constraint
domain, t2 mon
# soft deadlines
domain, t1 ends-by mon 3pm 10
domain, t2 ends-by mon 3pm 10
```

1.2.2 Get hard_constraint

```
# binary constraints
constraint, ⟨t1⟩ before ⟨t2⟩          # t1 ends when or before t2 starts
constraint, ⟨t1⟩ after ⟨t2⟩           # t1 starts after or when t2 ends
constraint, ⟨t1⟩ same-day ⟨t2⟩        # t1 and t2 are scheduled on the same day
constraint, ⟨t1⟩ starts-at ⟨t2⟩       # t1 starts exactly when t2 ends

# hard domain constraints
domain, ⟨t⟩ ⟨day⟩                      # t starts on given day at any time
domain, ⟨t⟩ ⟨time⟩                     # t starts at given time on any day
domain, ⟨t⟩ starts-before ⟨day⟩ ⟨time⟩          # at or before given time
domain, ⟨t⟩ starts-after ⟨day⟩ ⟨time⟩           # at or after given time
domain, ⟨t⟩ ends-before ⟨day⟩ ⟨time⟩            # at or before given time
domain, ⟨t⟩ ends-after ⟨day⟩ ⟨time⟩             # at or after given time
domain, ⟨t⟩ starts-in ⟨day⟩ ⟨time⟩-⟨day⟩ ⟨time⟩   # day-time range
domain, ⟨t⟩ ends-in ⟨day⟩ ⟨time⟩-⟨day⟩ ⟨time⟩     # day-time range
domain, ⟨t⟩ starts-before ⟨time⟩       # at or after time on any day
domain, ⟨t⟩ ends-before ⟨time⟩         # at or before time on any day
domain, ⟨t⟩ starts-after ⟨time⟩        # at or after time on any day
domain, ⟨t⟩ ends-after ⟨time⟩          # at or after time on any day
```

hard_constraint: [Constraint((scope), conditon),
Constraint(scope), condition),
.....]

```
# two tasks with two binary constraints and soft deadlines
task, t1 3
task, t2 4
# two binary constraints
constraint, t1 before t2
constraint, t1 same-day t2
# domain constraint
domain, t2 mon
# soft deadlines
domain, t1 ends-by mon 3pm 10
domain, t2 ends-by mon 3pm 10
```

- binary constraint

if “binary” in line and “constraints” in line:
对于后面的行开始开始检查，当遇到下一个#时停止
scope = (line[1] , line[3])
if line[2] == 'before':
condition = lambda x,y: x.end_time <= y.start_time
hard_constraint.append(Constraint(scope,condition))
if line[2] == 'same-day':
...
...

```
# two tasks with two binary constraints and soft deadlines
task, t1 3
task, t2 4
# two binary constraints
constraint, t1 before t2
constraint, t1 same-day t2
# domain constraint
domain, t2 mon
# soft deadlines
domain, t1 ends-by mon 3pm 10
domain, t2 ends-by mon 3pm 10
```

- domain constraint

if “domain” in line :

对于后面的行开始开始检查，当遇到下一个#时停止

```
if line[2] in day:
    scope = (line[1],)
    condition = lambda x: x.start_time.day ==day[line[2]]
    hard_constraint.append(Constraint(scope,condition))
if line[2] in time:
    ...
    ...
```

1.2.3 Get soft_constraint and soft_cost

```
# soft deadline constraints
domain, ⟨t⟩ ends-by ⟨day⟩ ⟨time⟩ ⟨cost⟩ # cost per hour of missing deadline
```

soft_domain: {t1: time, t2: time }
soft_cost: {t1: cost, t2: cost}

OR

soft_domain: {t1: (time，cost) , t2: (time, cost) }

1.3 构建完成

csp = New_CSP(domain,hard_constraint,soft_constraint,soft_cost)

2 对csp问题搜索求解

AIpython 的例子
searcher1 = Searcher(Search_from_CSP(csp1))
见代码

我们需要做的:

- 搜索过程中使用Arc Consistency：

  新建Search_with_AC_from_Cost_CSP继承Search_with_AC_from_CSP

- 搜索最优解(cost 最小): 使用greedy Search,

  将AStarSearcher 中的cost去掉

  在Search_with_AC_from_Cost_CSP中增加heuristic 函数

2.1 新建Search_with_AC_from_Cost_CSP

```
class Search_with_AC_from_Cost_CSP(Search_with_AC_from_CSP):
    def __init__(self,csp):
        super().__init__(csp)
        self.soft_cons = csp.soft_constraints
        self.soft_cost = soft_cost
```

2.2 搜索最优解

2.2.1 将Asearcher改成Greedy Search

```
class AStarSearcher(Searcher):
    """returns a searcher for a problem.
    Paths can be found by repeatedly calling search().
    """

    def __init__(self, problem):
        super().__init__(problem)

    def initialize_frontier(self):
        self.frontier = FrontierPQ()

    def empty_frontier(self):
        return self.frontier.empty()

    def add_to_frontier(self,path):
        """add path to the frontier with the appropriate cost"""

        value = path.cost+self.problem.heuristic(path.end())
        self.frontier.add(path, value)
```

2.2.2 在Search_with_AC_from_Cost_CSP中增加heuristic 函数

计算每个变量里每个value的cost
取每个变量里最小的cost
将所有变量的cost加起来

最终实现：

```
problem = Search_with_AC_from_Cost_CSP(csp)
search = AStarSearcher(problem).search()
```

输出：见代码

Coding Style(5分)
每个function加注释
开头写描述

THANKS

更多UNSW CSE课程，请咨询微信