Lab 5 Constraint Satisfaction Problems

SUSTC

Backtracking Search

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
     if value is consistent with assignment then
        add {var = value} to assignment
        inferences ← INFERENCE(csp, var, value)
        if inferences ≠ failure then
           add inferences to assignment
           result ← BACKTRACK(assignment, csp)
           if result ≠ failure then
              return result
     remove {var = value} and inferences from assignment
  return failure
```

Backtracking Search

```
def backtracking_search(csp,
                        select_unassigned_variable=first_unassigned_variable,
                        order_domain_values=unordered_domain_values,
                        inference=no_inference):
    """[Figure 6.5]
    """

    def backtrack(assignment):
        if len(assignment) == len(csp.variables):
            return assignment
        var = select_unassigned_variable(assignment, csp)
        for value in order_domain_values(var, assignment, csp):
            if 0 == csp.nconflicts(var, value, assignment):
                csp.assign(var, value, assignment)
                removals = csp.suppose(var, value)
                if inference(csp, var, value, assignment, removals):
                    result = backtrack(assignment)
                    if result is not None:
                        return result
                csp.restore(removals)
        csp.unassign(var, assignment)
        return None

    result = backtrack({})
    assert result is None or csp.goal_test(result)
    return result
```

Variable, Value ordering and Inference

-Used in Backtracking search

Heuristics	Methods		
Variable Order	First unassigned	MRV	-
Value Order	Default order	LCV	MC
Inference	No inference	Forward checking	MAC

Variable, Value ordering and Inference

-Used in Backtracking search

- Backtracking
  - Find the legal value for the variable
- MRV
  - Pick the variable with fewest legal values
- LCV
  - Pick the value with least constraint
- MC
  - Pick the value with minimal conflicts
- Forward checking
  - Rule out the illegal value
- MAC
  - Maintain arc consistence

Coding Example

```
from csp import (MapColoringCSP, min_conflicts, backtracking_search, mrv, forward_checking, mac)
import time
import networkx as nx
import matplotlib.pyplot as plt
```

- Variable ordering

```
def mrv(assignment, csp):
    "Minimum-remaining-values heuristic."
    return argmin_random_tie(
        [v for v in csp.variables if v not in assignment],
        key=lambda var: num_legal_values(csp, var, assignment))
```

- Value ordering

```
def lcv(var, assignment, csp):
    "Least-constraining-values heuristic."
    return sorted(csp.choices(var),
                  key=lambda val: csp.nconflicts(var, val, assignment))
```

Coding Example

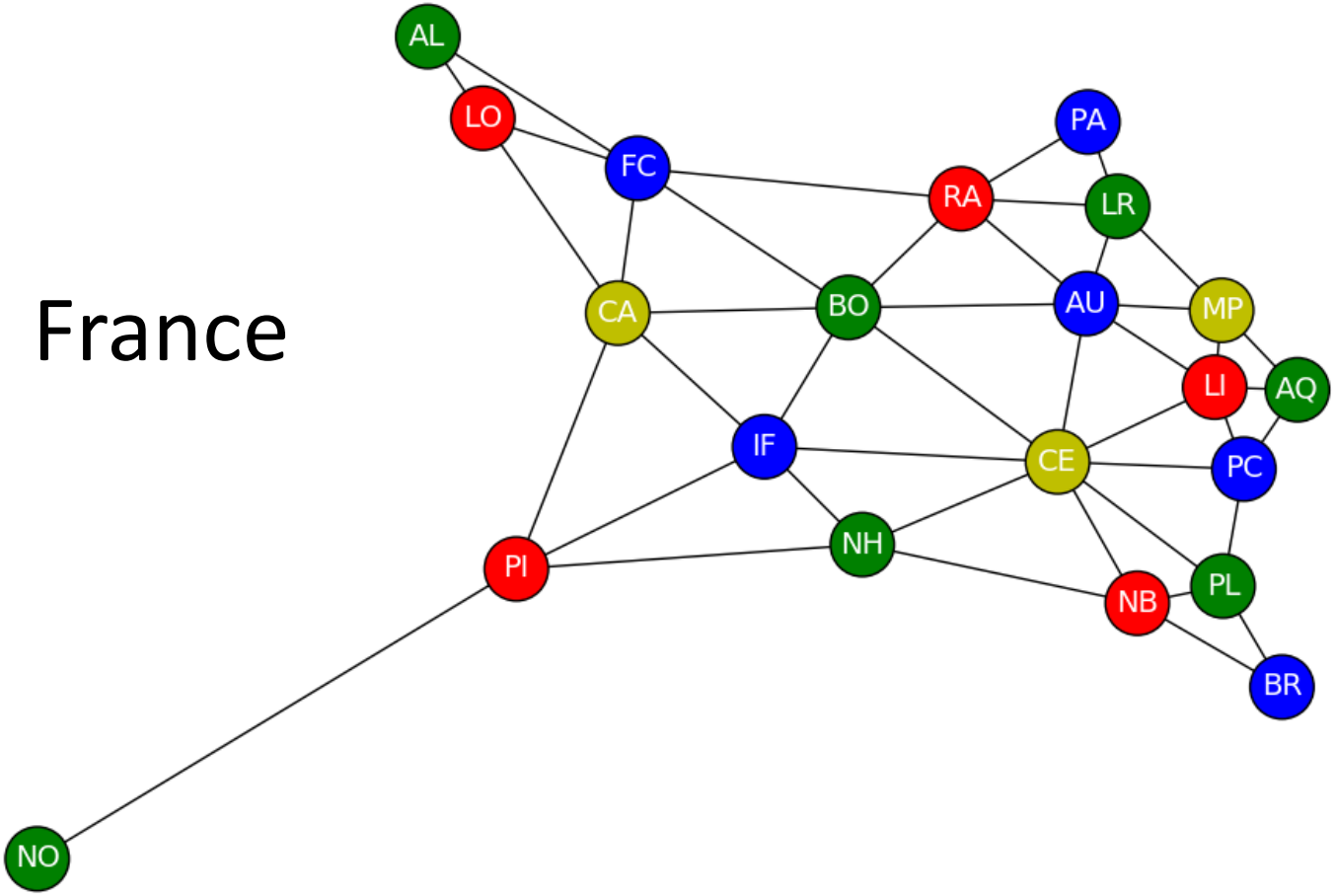
- Inference

```
def forward_checking(csp, var, value, assignment, removals):
    "Prune neighbor values inconsistent with var=value."
    for B in csp.neighbors[var]:
        if B not in assignment:
            for b in csp.curr_domains[B][:]:
                if not csp.constraints(var, value, B, b):
                    csp.prune(B, b, removals)
            if not csp.curr_domains[B]:
                return False
    return True


def mac(csp, var, value, assignment, removals):
    "Maintain arc consistency."
    return AC3(csp, [(X, var) for X in csp.neighbors[var]], removals)
```

Map-Coloring Problems

France



Coding Example

```
france = MapColoringCSP(list('RGBYK'),
                        """AL: LO FC; AQ: MP LI PC; AU: LI CE BO RA LR MP; BO: CE IF CA FC RA
        AU; BR: NB PL; CA: IF PI LO FC BO; CE: PL NB NH IF BO AU LI PC; FC: BO
        CA LO AL RA; IF: NH PI CA BO CE; LI: PC CE AU MP AQ; LO: CA AL FC; LR:
        MP AU RA PA; MP: AQ LI AU LR; NB: NH CE PL BR; NH: PI IF CE NB; NO:
        PI; PA: LR RA; PC: PL CE LI AQ; PI: NH NO CA IF; PL: BR NB CE PC; RA:
        AU BO FC PA LR""")

pstart = time.clock()

mymap=france
for itm in range(10):

    #mysol=min_conflicts(mymap)
    #mysol=backtracking_search(mymap)
    #mysol=backtracking_search(mymap,select_unassigned_variable=mrv)
    mysol=backtracking_search(mymap,inference=forward_checking)
    #mysol=backtracking_search(mymap,inference=mac)
    #mysol=backtracking_search(mymap,select_unassigned_variable=mrv,
                              #inference=forward_checking)

pend = time.clock()
ptime=(pend-pstart)/10
print("The running time: %s" %(ptime))
print(mysol)
```

Search CSP and record running time

Network Visualization

```
mystr=list(mymap.neighbors.items())

G = nx.Graph()
for i in range(len(mysol)):
    myloc=mystr[i][0]
    mynb =mystr[i][1]
    for j in range(len(mynb)):
        G.add_edge(myloc,mynb[j])

#assign colors to the node in correct order
node_colors=[]
for i in G.node:
    node_colors.append(mysol[i])

nx.draw_spectral(G,with_labels = True,font_size = 14,node_size = 900,
        font_color = 'w', node_color = node_colors)
plt.show()
```