

```

import networkx as nx
import matplotlib.pyplot as plt

class ConstraintSatisfactionProblem:
    def __init__(self, variables, domains, constraints):
        self.variables = variables
        self.domains = domains
        self.constraints = constraints

    def is_consistent(self, variable, value, assignment):
        return all(
            constraint(variable, value, assignment)
            for constraint in self.constraints.get(variable, []))

    def backtrack(self, assignment):
        if len(assignment) == len(self.variables):
            return assignment

        var = self.select_unassigned_variable(assignment)

        for value in self.order_domain_values(var, assignment):
            if self.is_consistent(var, value, assignment):
                assignment[var] = value
                result = self.backtrack(assignment)
                if result is not None:
                    return result
                assignment.pop(var)

        return None

    def select_unassigned_variable(self, assignment):
        return next(v for v in self.variables if v not in assignment)

    def order_domain_values(self, variable, assignment):
        return self.domains[variable]

variables = ['Exam1', 'Exam2', 'Exam3', 'Exam4']

proctors = {
    'P1': {'available': ['Morning'], 'skills': ['Math', 'Physics']},
    'P2': {'available': ['Morning', 'Afternoon'], 'skills': ['CS', 'Math']},
    'P3': {'available': ['Afternoon'], 'skills': ['Math', 'CS']},
    'P4': {'available': ['Morning', 'Afternoon'], 'skills': ['Physics', 'CS']}
}

exam_details = {
    'Exam1': {'time': 'Morning', 'subject': 'Math'},
    'Exam2': {'time': 'Morning', 'subject': 'CS'},
    'Exam3': {'time': 'Afternoon', 'subject': 'Math'},
    'Exam4': {'time': 'Afternoon', 'subject': 'Physics'}
}

domains = {exam: list(proctors.keys()) for exam in variables}

def create_constraints():
    constraints = {}

    for exam in variables:
        constraints[exam] = []

    def skill_time_constraint(var, val, ass, exam=exam):
        return (
            exam_details[exam]['time'] in proctors[val]['available']
            and exam_details[exam]['subject'] in proctors[val]['skills']
        )

    constraints[exam].append(skill_time_constraint)

    def no_clash_constraint(var, val, ass, exam=exam):
        for assigned_exam, assigned_proctor in ass.items():
            if assigned_proctor == val:
                if exam_details[assigned_exam]['time'] == exam_details[exam]['time']:
                    return False
    constraints[exam].append(no_clash_constraint)
}

```

```

        return True

    constraints[exam].append(no_clash_constraint)

    return constraints

constraints = create_constraints()

csp = ConstraintSatisfactionProblem(variables, domains, constraints)
solution = csp.backtrack({})

print("Proctor Assignment:")
for exam, proctor in solution.items():
    print(f"{exam}: {proctor}")

G = nx.Graph()

for exam in variables:
    G.add_node(exam)

G.add_edge('Exam1', 'Exam2')
G.add_edge('Exam2', 'Exam3')
G.add_edge('Exam3', 'Exam4')
G.add_edge('Exam4', 'Exam1')
G.add_edge('Exam1', 'Exam3')

pos = {
    'Exam1': (0, 1),
    'Exam2': (1, 1),
    'Exam3': (1, 0),
    'Exam4': (0, 0)
}

labels = {node: f"{node}\n{solution[node]}" for node in G.nodes()}

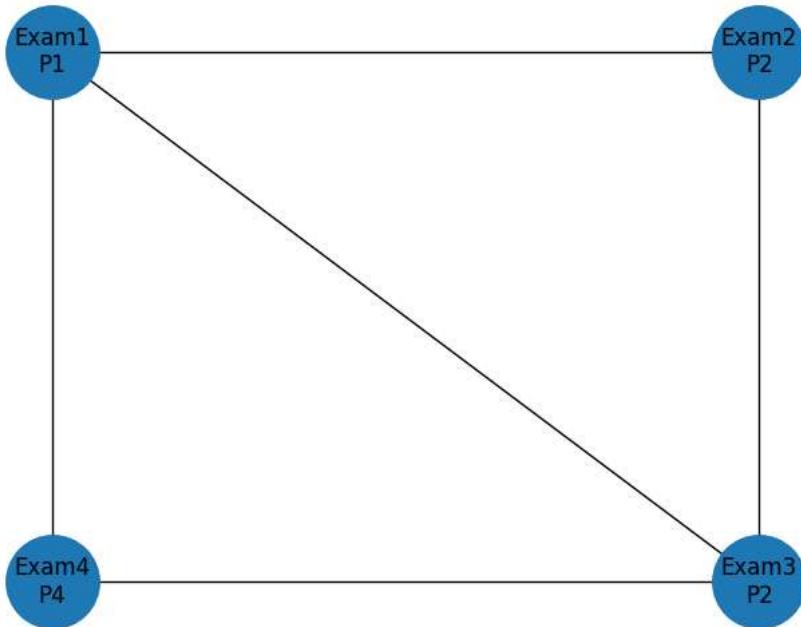
plt.figure()
nx.draw(G, pos, with_labels=False, node_size=2500)
nx.draw_networkx_labels(G, pos, labels)

plt.title("Proctor Assignment CSP Graph")
plt.show()

```

Proctor Assignment:
 Exam1: P1
 Exam2: P2
 Exam3: P2
 Exam4: P4

Proctor Assignment CSP Graph



Start coding or generate with AI.