Mini Project 1: Where there's smoke, there's fire!

Hand-in Instructions

- The assignment is due by February 18, 2026 at midnight.
- Please submit to Gradescope (accessible via Canvas). There will be **two** assignments on Gradescope which you will need to submit to: one for your written answers, and one for code upload.
- Your written answers should be in a single PDF file, ideally produced with LaTeX.
- The goal here is learning. You can work with other students, but be sure that, in the end, the solutions you submit were your work and that you understand them completely.
- You may use LLMs to help with this assignment; note, though, that we've tried these questions in several LLMs and they often produce incorrect answers, so the correctness is up to you!

1 Smoke and Fire

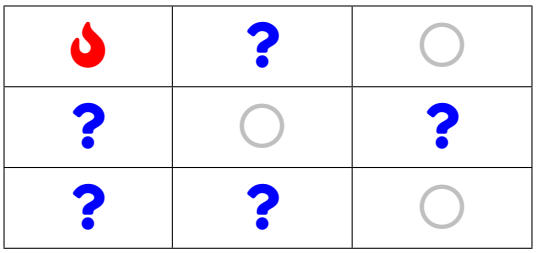
We will address an inference problem that is kind of like Minesweeper, by formulating it as a discrete constraint satisfaction problem (CSP) and solving it using a downloadable solver.

Our environment is a 2D rectangular grid. Every cell in the grid may have fire in it, may have smoke but no fire, or may be clear.

We will make some observations of the states of some cells in the grid. And we know some "physics" rules about how smoke and fire are geographically related (all within the same moment in time):

1. There is smoke at a location only if there is a fire in at at least one of the 4 adjacent locations.
2. If there is a fire at some location, then there is smoke *or* fire at every adjacent location.

Consider the following grid (circles indicate we have observed that cell to be clear):



Take a moment to run your human inference engine: which unknown values in the grid above can be determined?

1. (10 points) Write a Python program, using the `python_constraint` package, that can take in as input an arbitrary rectangular array with characters `'F'`, `'S'`, `'C'`, `'U'`, indicating fire, smoke, clear, and unknown, and produce a new array of characters of the same shape, in which any `'U'` character in a cell whose contents can be unambiguously determined from the input and the physics rules is replaced by a character indicating the inferred contents.

   What output does your code generate for the input below?

```
grid = [
['C', 'C', 'C', 'C', 'C', 'C', 'C'],
['C', 'C', 'S', 'U', 'S', 'C', 'C'],
['C', 'U', 'F', 'U', 'F', 'U', 'C'],
['C', 'U', 'U', 'U', 'S', 'C', 'C'],
['C', 'C', 'U', 'U', 'C', 'C', 'C'],
['C', 'U', 'C', 'U', 'U', 'C', 'C'],
['C', 'C', 'C', 'C', 'C', 'C', 'C']
]
```

Solution:

```
from constraint import Problem
from itertools import product

# States: F=fire, S=smoke, C=clear
# Output will use U if a cell differs across solutions.

grid = [
    ['C', 'C', 'C', 'C', 'C', 'C', 'C'],
    ['C', 'C', 'S', 'U', 'S', 'C', 'C'],
    ['C', 'U', 'F', 'U', 'F', 'U', 'C'],
    ['C', 'U', 'U', 'U', 'S', 'C', 'C'],
    ['C', 'C', 'U', 'U', 'C', 'C', 'C'],
    ['C', 'U', 'C', 'U', 'U', 'C', 'C'],
    ['C', 'C', 'C', 'C', 'C', 'C', 'C'],
]

rows, cols = len(grid), len(grid[0])
def vname(r, c): return f"cell_{r}_{c}"

def local_rule(*vals):
    cell = vals[0]
    nbrs = vals[1:]

    if cell == "F":
        return all(v in ("S", "F") for v in nbrs)
    if cell == "C":
        return all(v in ("S", "C") for v in nbrs)
    if cell == "S":
        return any(v == "F" for v in nbrs)
    return True

problem = Problem()

# Unknowns are variables over {F,S,C}. Do not allow "U" as an assigned value.
for r in range(rows):
    for c in range(cols):
        name = vname(r, c)
        if grid[r][c] == "U":
            problem.addVariable(name, ["F", "S", "C"])
        else:
```

```
            problem.addVariable(name, [grid[r][c]])

# Local constraints (4-neighborhood)
for r in range(rows):
    for c in range(cols):
        center = vname(r, c)
        nbrs = []
        for dr, dc in ((0, 1), (1, 0), (0, -1), (-1, 0)):
            rr, cc = r + dr, c + dc
            if 0 <= rr < rows and 0 <= cc < cols:
                nbrs.append(vname(rr, cc))
        problem.addConstraint(local_rule, [center] + nbrs)

if __name__ == "__main__":
    it = problem.getSolutionIter()
    first = next(it, None)

    if first is None:
        print("No solutions")
        raise SystemExit(1)

    consensus = [[first[vname(r, c)] for c in range(cols)] for r in range(rows)]
    ambiguous = [[False] * cols for _ in range(rows)]
    count = 1

    for sol in it:
        count += 1
        for r in range(rows):
            for c in range(cols):
                if not ambiguous[r][c] and sol[vname(r, c)] != consensus[r][c]:
                    ambiguous[r][c] = True

    final_grid = [
        ["U" if ambiguous[r][c] else consensus[r][c] for c in range(cols)]
        for r in range(rows)
    ]

    print(f"solutions: {count}")
    for row in final_grid:
        print(row)
```

Solution:

```
grid=[
['C', 'C', 'C', 'C', 'C', 'C', 'C'],
['C', 'C', 'S', 'U', 'S', 'C', 'C'],
['C', 'S', 'F', 'U', 'F', 'S', 'C'],
['C', 'U', 'U', 'U', 'S', 'C', 'C'],
['C', 'C', 'U', 'U', 'C', 'C', 'C'],
['C', 'C', 'C', 'C', 'C', 'C', 'C'],
```

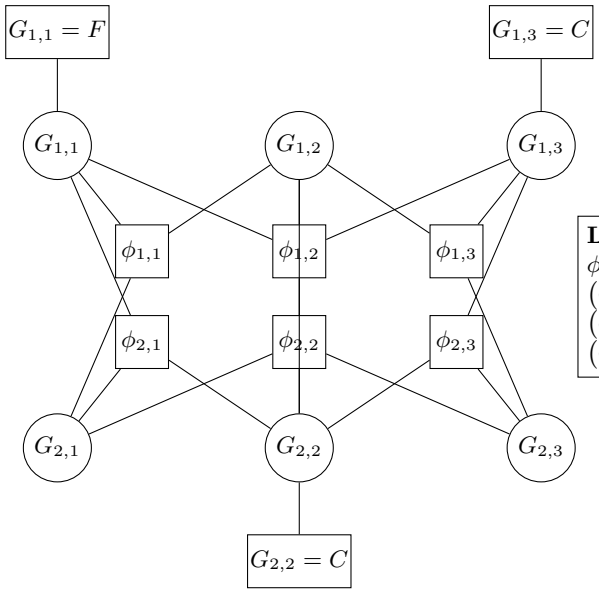
```
['C', 'C', 'C', 'C', 'C', 'C', 'C']
]
```

2. (15 points) How did you decide whether to leave a cell as `'U'` in your output?

Solution: If there exist two satisfying assignments consistent with the input observations and the physics rules that give different values (`F`, `S`, or `C`) to that cell, then leave it as `U`. Otherwise, if all satisfying assignments agree, output that common value.

3. (15 points) Draw the constraint graph for the 2 x 3 top cells in the example provided on page 1 ($[FUC; UCU]$). Remember the unary factors for the observations.

Solution:



Legend. For cell (r, c) with orthogonal neighbors $N(r, c)$,
$\phi_{r,c}\big(G_{r,c}, (G_n)_{n \in N(r,c)}\big) = 1 \iff$
$\big(G_{r,c} = F \Rightarrow \forall n \in N(r, c) : G_n \in \{F, S\}\big) \wedge$
$\big(G_{r,c} = C \Rightarrow \forall n \in N(r, c) : G_n \in \{C, S\}\big) \wedge$
$\big(G_{r,c} = S \Rightarrow \exists n \in N(r, c) : G_n = F\big)$.

4. (10 points) Assume we start with a partial assignment of all the variables with given values determined. Consider the variable ordering of starting in the upper left-hand corner, filling out the top row from left to right, then the next row, etc. Is it possible to find a satisfying assignment to our first example problem on page 1, using forward checking, without backtracking, **for all value orderings?** If not, explain. If yes, what assignment is found?

Note that The version of forward checking in the book assumes that we have binary constraints. You can use a variation on forward checking in which, when you make an assignment to a variable, you reduce the domains of any other variables that share a constraint as much as possible (but don't do any further propagation).

Solution: We will assume we have done forward checking to reduce domains
already, based on the assigned variables.

- First one only has 'S' left in its domain

- Next one only has 'S'
- Next one only has 'C'
- Next one can be 'S' or 'F' or 'C'
- Last one can be 'S' or 'C'

So, when we're at the bottom-left node, we could pick 'S'.

Then, if we do that, we discover that there are no satisfying assignments for the last node.

So, we have to backtrack.

5. (10 points) If you had a large problem in which you knew that a whole row of cells across the center was clear, would that affect the worst-case complexity of solving an $n \times n$ problem by backtracking search? If so, how? Would the worst-case complexity class change?

Solution: Yes. Knowing that the entire middle row is clear acts like a separator, and it removes any fire/smoke dependencies across that row. Then, a backtracking solver can treat the cells above and below as two independent subproblems and solve them separately. This reduces the effective search effort by a constant-factor in the exponent (approximately replacing one $O(|D|^{(n^2)})$ search by two searches of size about $O(|D|^{(n^2/2)})$), but it does not change the worst-case asymptotic complexity; i.e. the runtime remains exponential in n^2. Therefore, the worst-case complexity class would not change.

2 Teaching Yourself

The example above was relatively simple to represent and solve. In contrast, most real-world problems can be modeled in several different ways, and there are many reasonable constraints one might impose on a model to obtain a solution. In practice, some constraints are more effective than others and can help us reach solutions much more quickly. Moreover, certain constraints allow more flexibility than others. In this problem, you will construct a model and describe the constraints used to represent the process of matching students with tutors.

A bit more formally, imagine we have a set T of tutors, set S of students, set H of hours in the week, and set C of classes. Each tutor $t \in T$ is qualified to teach a subset of classes $C_t \subseteq C$ and is available during a subset of hours $H_t \subseteq H$. Each student $s \in S$ requires help in a subset of classes $C_s \subseteq C$ and is available to receive tutoring during a subset of hours $H_s \subseteq H$.

6. (15 points) Your goal is to find a valid assignment of tutors to students. At any hour $h \in H$, a student may be assigned to at most one tutor for at most one class, and a tutor may teach at most one student for at most one class. Assignments must respect both the tutor qualifications and the availability constraints of tutors and students.

   Define a constraint satisfaction problem that models this real-world situation. Specify the variables, the domains of the variables, and the constraints in some formal notation (math or pseudocode).

Solution:

Variables. For each student–hour pair and tutor–hour pair, define

$$\forall (s,h) \in S \times H: \quad X_{sh} \in (T \times C) \cup \varnothing, \qquad \forall (t,h) \in T \times H: \quad Y_{th} \in (S \times C) \cup \varnothing.$$

Domains. For all $s \in S$ and $h \in H$, restrict

$$X_{sh} \in \{(t, c) \in T \times C : \ h \in H_s \cap H_t, \ c \in C_s \cap C_t\} \cup \{\varnothing\}.$$

For all $t \in T$ and $h \in H$, restrict

$$Y_{th} \in \{(s, c) \in S \times C : \ h \in H_s \cap H_t, \ c \in C_s \cap C_t\} \cup \{\varnothing\}.$$

Constraint. For all $s \in S$, $t \in T$, $h \in H$, and $c \in C$,

$$X_{sh} = (t, c) \iff Y_{th} = (s, c).$$

7. (15 points) The model above can lead to many solutions that wouldn't be practical in the real world. What are some other constraints that you might want to add to this model to make it more realistic and increase the utility of the resulting solutions? List 3 considerations that this model does not account for, and how the model could be modified and constraints added to account for them. For each one, (1) explain in English the weakness of the current formulation that you are addressing, (2) explain in English the constraint that you propose to add, and (3) provide a formal description of the constraint that's compatible with your formulation above.

Solution:

Here are a few examples that would earn full credit, though there are more. For a tutor t and hour h, write

$$\text{Uses}(t, h) \ :\iff \ \exists s \in S, \exists c \in C : \ X_{sh} = (t, c).$$

1. **Maximum number of hours per tutor per week.** Fix a cap $K_t \in \{0, 1, \ldots, |H|\}$ for each tutor t. Enforce:

$$\forall t \in T, \ \forall B \subseteq H \ (|B| = K_t + 1) : \quad \neg\Big(\bigwedge_{h \in B} \text{Uses}(t, h)\Big).$$

2. **Each student gets at least a few hours.** Fix a minimum $M_s \in \{0, 1, \ldots, |H|\}$ for each student s. Enforce:

$$\forall s \in S, \ \forall B \subseteq H \ (|B| = |H| - M_s + 1) : \quad \neg\Big(\bigwedge_{h \in B} (X_{sh} = \varnothing)\Big).$$

3. **Allow small groups (same tutor, same class, same hour).** Fix a group size limit $g \in \mathbb{N}$. For each $(t, c, h) \in T \times C \times H$, introduce g "slots"

$$Z^{(1)}_{tch}, \ldots, Z^{(g)}_{tch} \in S \cup \{\varnothing\}.$$

Add (i) no duplicate non-empty slots:

$$\forall t, c, h, \ \forall i < j : \quad \big(Z^{(i)}_{tch} = Z^{(j)}_{tch}\big) \implies \big(Z^{(i)}_{tch} = \varnothing\big),$$

(ii) slots imply assignments:

$$\forall t, c, h, \forall i, \forall s \in S : \quad Z^{(i)}_{tch} = s \implies X_{sh} = (t, c),$$

(iii) assignments imply some slot is used:

$$\forall s, h, \forall t, c : \quad X_{sh} = (t, c) \implies \bigvee_{i=1}^{g} \big(Z^{(i)}_{tch} = s\big).$$

This allows up to g students per (t, c, h).

4. **Balance tutor load.** We can enforce both a per-tutor minimum and maximum number of hours per week. Fix $L_t, U_t \in \{0, 1, \ldots, |H|\}$ and add:

$$\forall t \in T,\ \forall B \subseteq H\ (|B| = U_t + 1): \quad \neg\Big(\bigwedge_{h \in B} \text{Uses}(t, h)\Big),$$

$$\forall t \in T,\ \forall B \subseteq H\ (|B| = |H| - L_t + 1): \quad \neg\Big(\bigwedge_{h \in B} \neg\text{Uses}(t, h)\Big).$$

Choosing L_t and U_t close forces loads to be similar.

5. **One tutor per student per class.** Introduce $U_{sc} \in T \cup \{\varnothing\}$ for each $(s, c) \in S \times C$ and enforce:

$$\forall s \in S, \forall c \in C, \forall h \in H, \forall t \in T: \quad X_{sh} = (t, c) \implies U_{sc} = t.$$

6. **One tutor per student across classes.** Introduce $U_s \in T \cup \{\varnothing\}$ for each $s \in S$ and enforce:

$$\forall s \in S, \forall h \in H, \forall t \in T, \forall c \in C: \quad X_{sh} = (t, c) \implies U_s = t.$$

7. **Need and quality based matching.** Let need : $S \to N$ and qual : $T \to Q$ be given score functions to finite sets representing student need and tutor quality, respectively. Define a set $R \subseteq N \times Q$ that represents valid quality and need matching for students and tutors. Enforce:

$$\forall s \in S, \forall h \in H, \forall t \in T, \forall c \in C: \quad X_{sh} = (t, c) \implies \big(\text{need}(s), \text{qual}(t)\big) \in R.$$

8. (4 points) In our original model, is it possible that there could be more tutors than students, yet the demand still could not be met? Explain in 1-2 sentences.

Solution: Yes. If the courses a tutor could teach do not align with the courses the students need help in, or the hours tutors are available do not align with those the students need help with, perfect solutions are impossible.

9. (3 points) In our original model, could there exist satisfying assignments where no one is paired? Explain in 1-2 sentences.

Solution: Yes. In the model above, it is a valid solution to not have anybody paired.

10. (3 points) How would you deal with a situation where there are not enough tutors to meet the demand (think outside the box)? Explain in 1-2 sentences.

Solution:

Here are a few ways that one could deal with not having enough tutors.

1. Increase pay for tutors to attract more.
2. Allow tutors to teach courses similar to ones they have taken.
3. Allow multiple students to learn from a tutor at a time.