SOLVING THE YASHI PUZZLE

WITH A SAT SOLVER

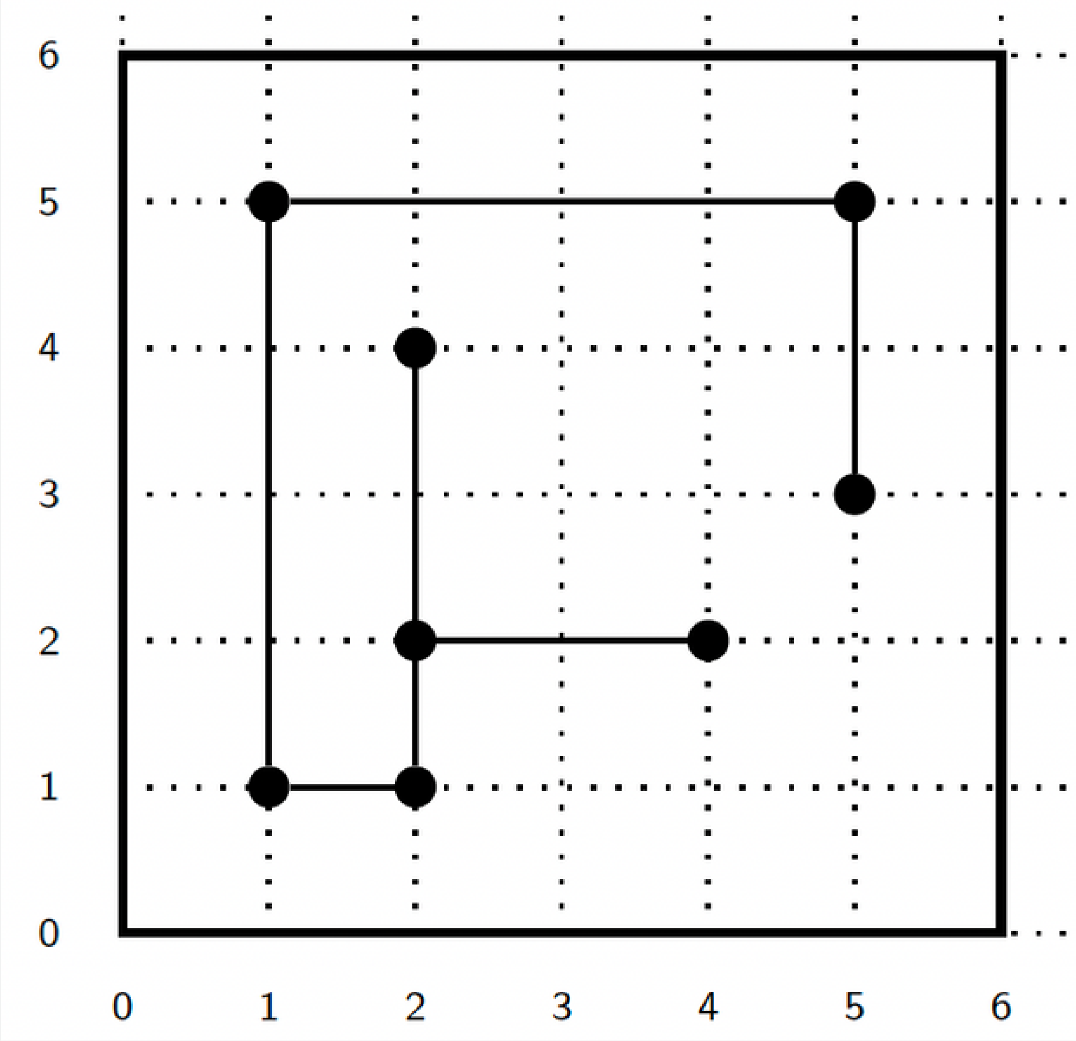
August 2023 | Università degli Studi di Padova

Joan Orellana Rios

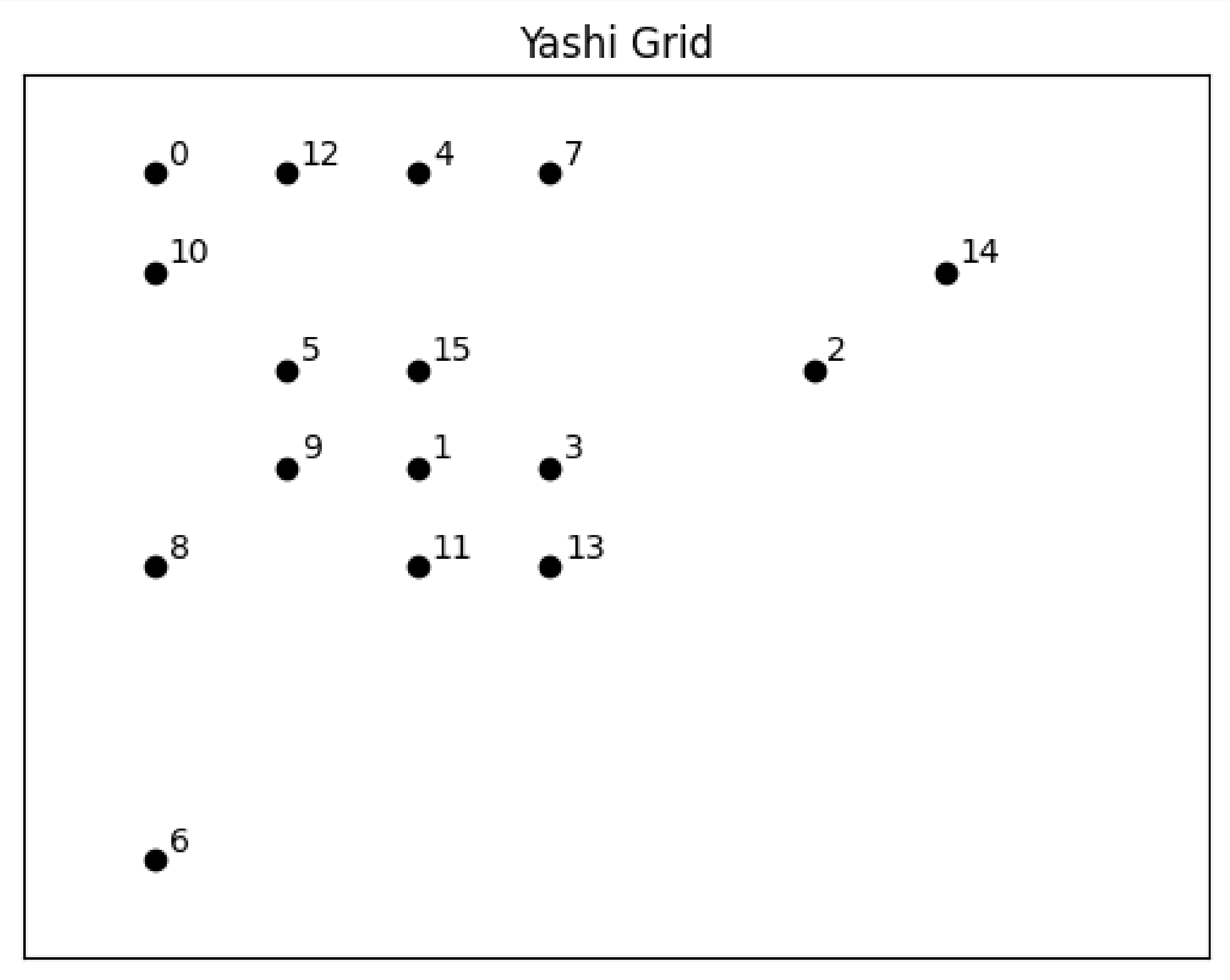
YASHI PUZZLE

RULES

- Every segment connects two and only two nodes.
- No two segments overlap.
- No two segments cross each other.
- The segments form a tree, i.e., they form a graph without cycles. Put differently still, for every two nodes a and b there is exactly one path between a and b.



PROBLEM



Solving the Yashi puzzle using a programming approach

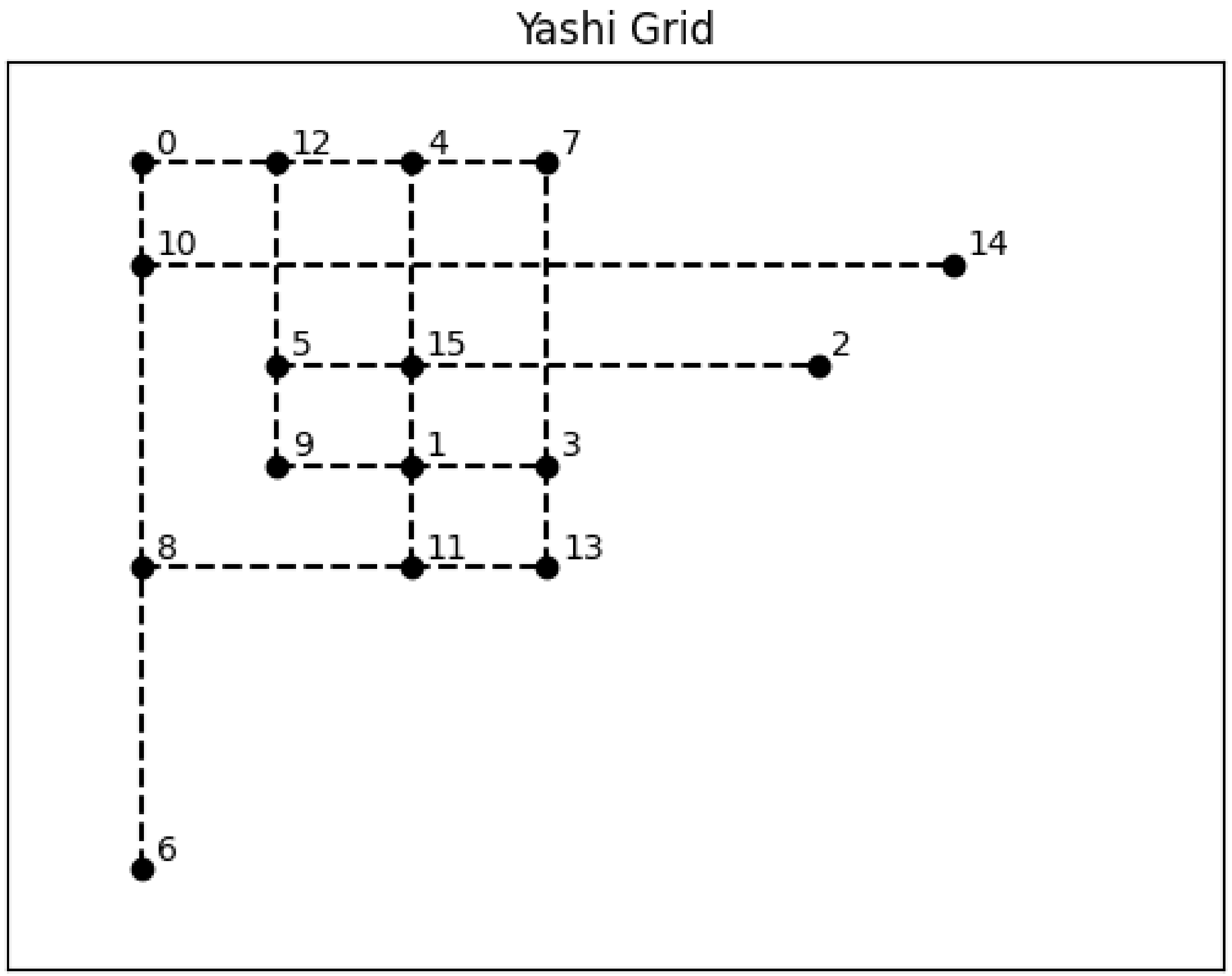
STEP 1

JOIN ALL THE POINTS

- Using only vertical and horizontal lines
- Non-overlapping segments

PROBLEMS

These segments still intersect each other and create cycles, which do not obey the rules.



STEP 2

DETECT INTERSECTIONS

No lines intersect each other

DETECT CYCLES

There are not two different path from point a to point b

FORM A SINGLE TREE

Make sure all the points are connected to the tree

DETECT INTERSECTIONS

Function: Check Intersections

Input: List of edges (point1,point2)

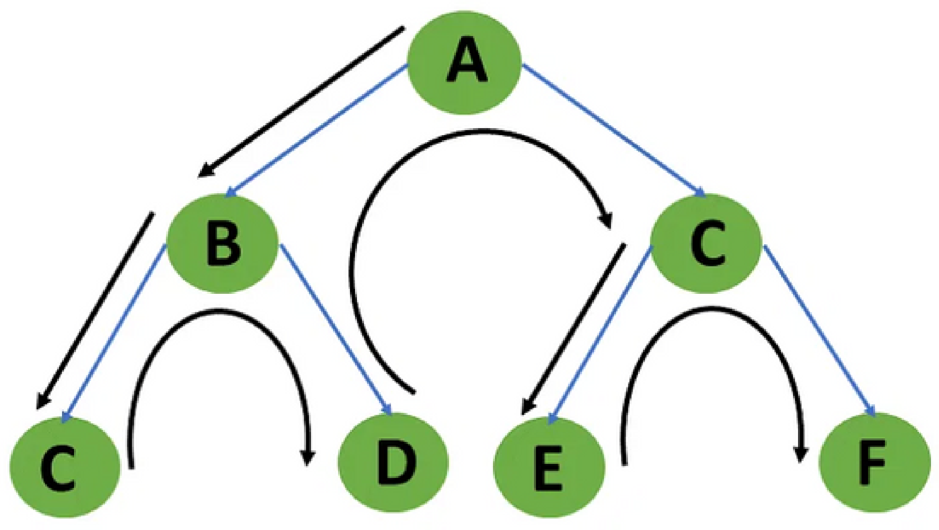
Output: List of edges that intersect

```
def check_intersections(lines):
    def check_intersection(line1, line2):
        (x1, y1), (x2, y2) = line1
        (x3, y3), (x4, y4) = line2

        return (min(x1, x2) < x3 < max(x1, x2) and min(y3, y4) < y1 < max(y3, y4))
            or (min(y1, y2) < y3 < max(y1, y2) and min(x3, x4) < x1 < max(x3, x4))

    intersection_segments = []
    for i in lines:
        for j in lines:
            if i != j:
                if check_intersection(i, j):
                    new_int = sorted((lines_idx[i], lines_idx[j]))
                    if new_int not in intersection_segments:
                        intersection_segments.append(new_int)

    return intersection_segments
```

Code of the function check_intersections

DETECT CYCLES

Function: Check Intersections
Input: Graph
Output: List of cycles

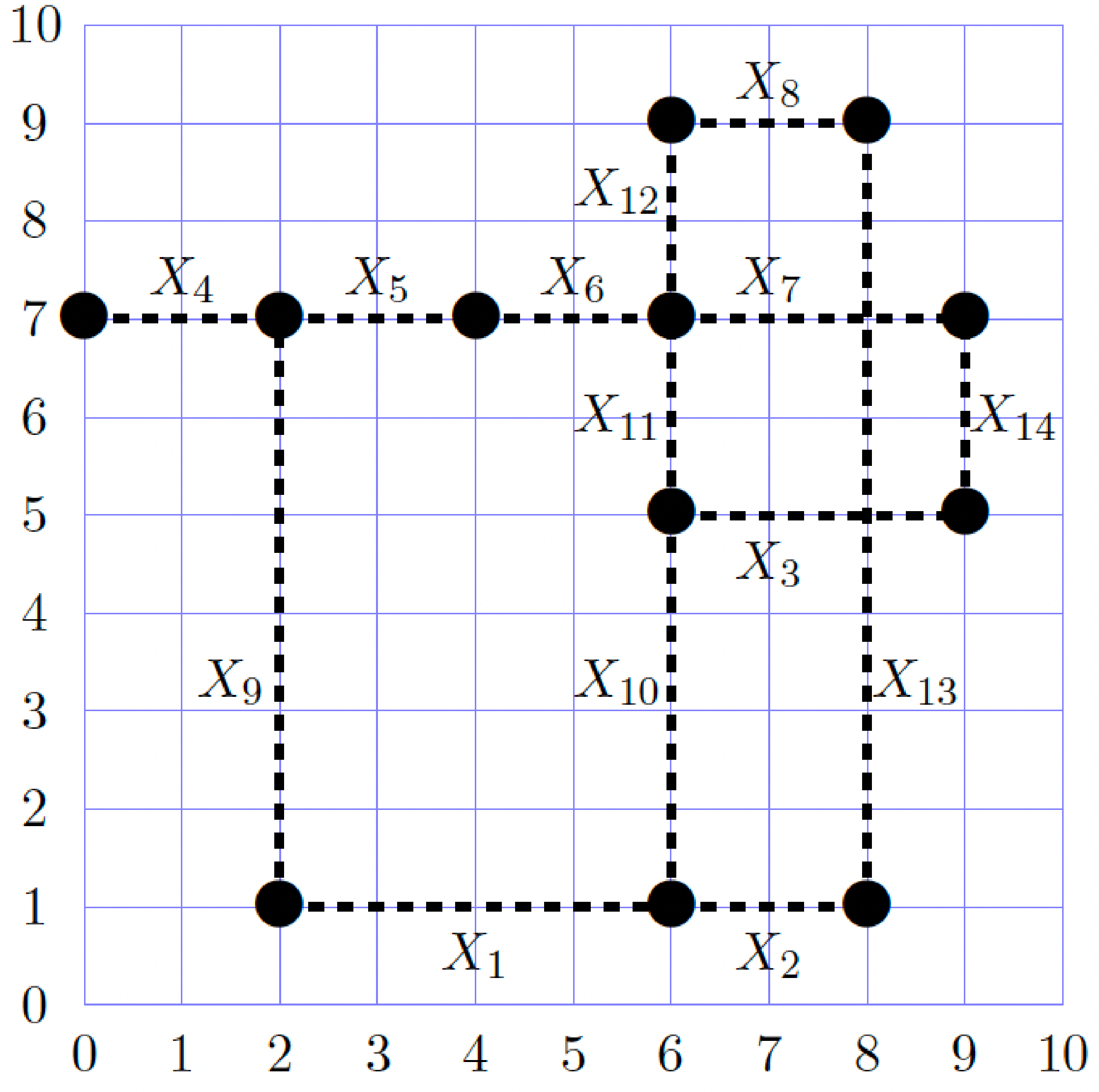


Depth-First Search algorithm

```
def find_all_cycles(graph):
    def dfs(current, start, path, path_nodes):
        nonlocal cycles, sort_cyc
        visited[current] = True
        path.append(current)
        for neighbor in graph[current]:
            if not visited[neighbor]:
                dfs(neighbor, start, path, path_nodes)
            elif neighbor == start and len(path) > 2:
                new_cycle = path[:]
                sorted_cycle = sorted(path[:])
                if sorted_cycle not in sort_cyc:
                    cycles.append(new_cycle)
                    sort_cyc.append(sorted_cycle)
        path.pop()
        if path_nodes != []:
            path_nodes.pop()
        visited[current] = False

    nodes = graph.keys()

    visited = {}
    for node in nodes:
        visited[node] = False
    cycles, sort_cyc = [], []

    for node in nodes:
        dfs(node, node, [], [])

    return cycles
```

Code of the function find_all_cycles

FORM A SINGLE TREE



No cut method

One-node cuts

$$\text{NO-CUT}_{\langle 2,1\rangle} \triangleq X_1 \vee X_9$$
$$\text{NO-CUT}_{\langle 6,1\rangle} \triangleq X_1 \vee X_2 \vee X_{10}$$
$$\text{NO-CUT}_{\langle 8,1\rangle} \triangleq X_2 \vee X_{13}$$

Two-node cuts

$$\text{NO-CUT}_{\{\langle 2,1\rangle,\langle 6,1\rangle\}} \triangleq X_2 \vee X_9 \vee X_{10}$$
$$\text{NO-CUT}_{\{\langle 6,1\rangle,\langle 8,1\rangle\}} \triangleq X_1 \vee X_{10} \vee X_{13}$$
$$\text{NO-CUT}_{\{\langle 0,7\rangle,\langle 2,7\rangle\}} \triangleq X_5 \vee X_9$$

Three-node cuts

$$\text{NO-CUT}_{\{\langle 0,7\rangle,\langle 2,1\rangle,\langle 2,7\rangle\}} \triangleq X_1 \vee X_5$$
$$\text{NO-CUT}_{\{\langle 0,7\rangle,\langle 2,7\rangle,\langle 4,7\rangle\}} \triangleq X_6 \vee X_9$$
$$\text{NO-CUT}_{\{\langle 2,1\rangle,\langle 6,1\rangle,\langle 8,1\rangle\}} \triangleq X_9 \vee X_{10} \vee X_{13}$$

Four-node cuts

$$\text{NO-CUT}_{\{\langle 0,7\rangle,\langle 2,1\rangle,\langle 2,7\rangle,\langle 4,7\rangle\}} \triangleq X_1 \vee X_6$$
$$\text{NO-CUT}_{\{\langle 2,1\rangle,\langle 6,1\rangle,\langle 8,1\rangle,\langle 6,5\rangle\}} \triangleq X_3 \vee X_9 \vee X_{11} \vee X_{13}$$

```
for i in range(1, len(lines) + 1):
    const = [-j if j != i else j for j in range(1, len(lines) + 1)]
    wcnf.append(lines_per_point[point])
```

Code to add one-node cut constraints with PySat

SOLVING USING PYSAT

INTERSECTIONS

$$(\neg X_3 \lor \neg X_{13}) \land (\neg X_7 \lor \neg X_{13})$$

Code

```
for cycle in cyc_path:
    const = [i * -1 for i in cycle]
    wcnf.append(const)
```

CYCLES

$$(\neg X_1 \lor \neg X_5 \lor \neg X_6 \lor \neg X_9 \lor \neg X_{10} \lor \neg X_{11})$$
$$\land\ (\neg X_1 \lor \neg X_3 \lor \neg X_5 \lor \neg X_6 \lor \neg X_7 \lor \neg X_9 \lor \neg X_{10} \lor \neg X_{14})$$
$$\land\ (\neg X_1 \lor \neg X_2 \lor \neg X_5 \lor \neg X_6 \lor \neg X_8 \lor \neg X_9 \lor \neg X_{12} \lor \neg X_{13})$$

Code

```
for intersect in intersections:
    const = [i * -1 for i in intersect]
    wcnf.append(const)
```

SINGLE TREE

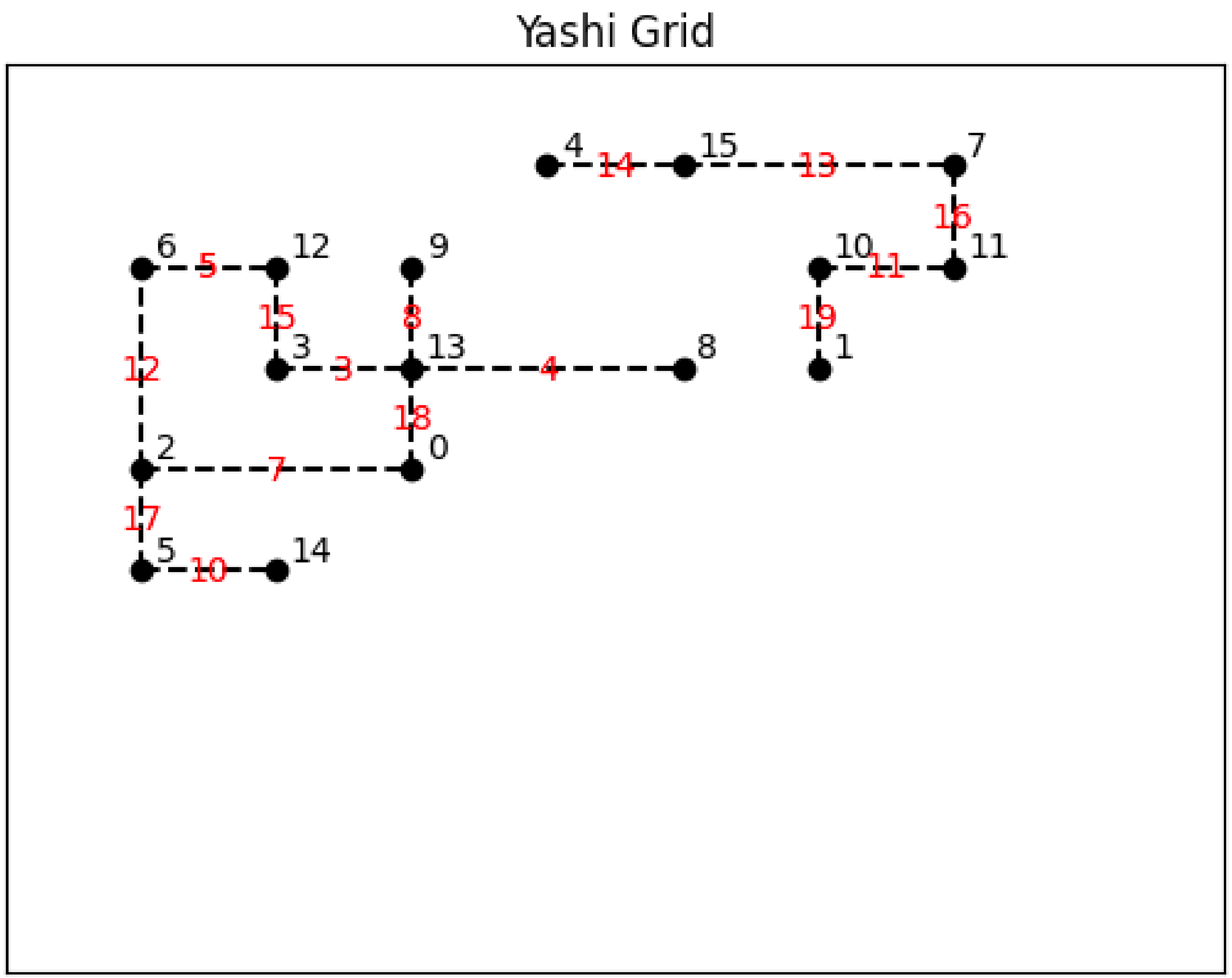
$$X_1 \lor X_9$$

$$X_1 \lor X_2 \lor X_{10}$$

Code

```
for i in range(1, len(lines) + 1):
    const = [-j if j != i else j
             for j in range(1, len(lines) + 1)]
    wcnf.append(lines_per_point[point])
```

PROBLEM: NO-CUT



SOLUTION: SMT SOLVER

Unlike traditional SAT (Boolean Satisfiability) solvers that deal with propositional logic only, SMT solvers can handle richer theories, such as integer arithmetic, real arithmetic, arrays, bit-vectors, and more.

NO-CUT → n-1 edges, where n is the number of nodes

USE OF THE Z3 LIBRARY

DEFINE VARIABLES

ADD CYCLES CONSTRAINTS

ADD INTERSECTIONS CONSTRAINTS

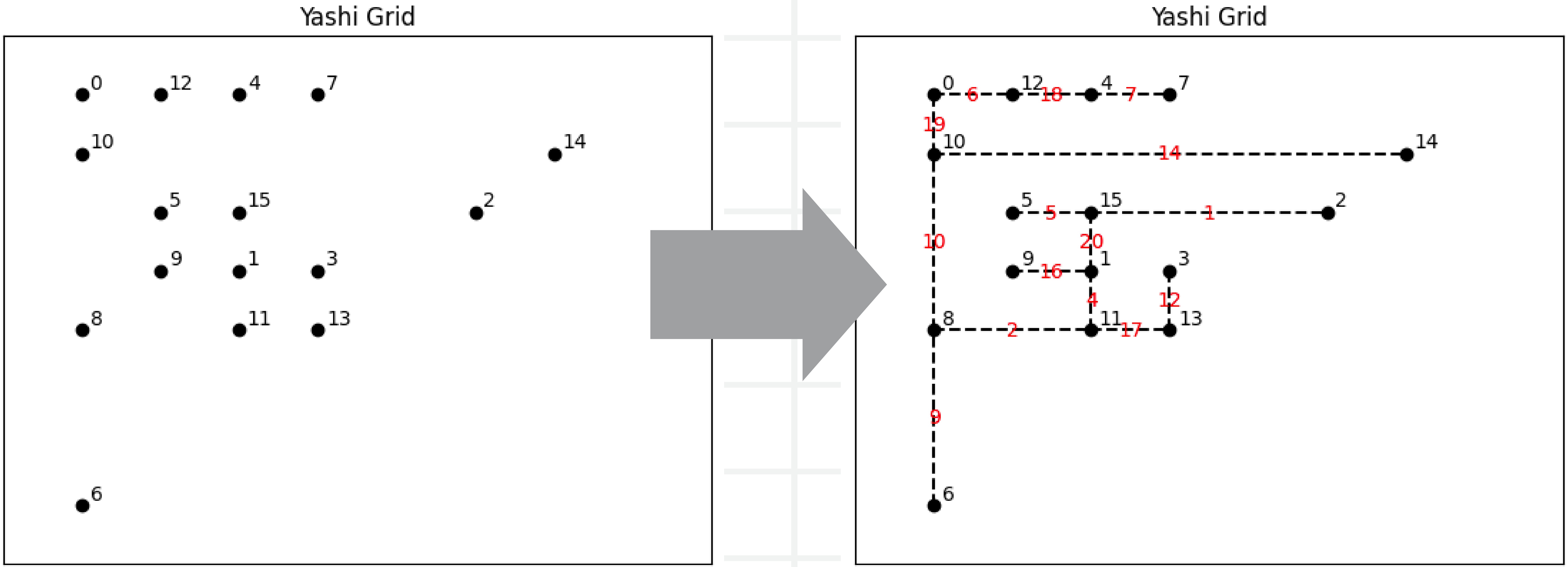
```
from z3 import *

s = Solver()

vars = [str(i) for i in range(1, len(lines) + 1)]
bool_vars = [Bool(v) for v in vars]

for cycle in cyc_lines:
    const = [Not(bool_vars[i-1]) for i in cycle]
    const = Or(tuple(const))
    s.add(const)

for intersect in intersections:
    const = [Not(bool_vars[i-1]) for i in intersect]
    const = Or(tuple(const))
    s.add(const)
```

```
# At least 'n' variables must be true
s.add(AtLeast(*bool_vars, 2*dim - 1))

# At most 'n' variables must be true
s.add(AtMost(*bool_vars, 2*dim - 1))

# Check satisfiability
result = s.check()

if result == sat:
    # Get the model and extract the values of the variables
    model = s.model()
    true_vars = [v for v in bool_vars if is_true(model.eval(v))]
    plot_model()
else:
    print('Not Satisfiable')
```

SINGLE TREE EXACTLY N-1 EDGES

Z3 Solver in Python

RESULT


Yashi Grid
0 12 4 7
10 14
5 15 2
9 1 3
8 11 13
6
Yashi Grid
0 6 12 18 4 7 7
19
10 14 14
5 5 15 1 2
10 20
9 16 1 3
4 12
8 2 11 17 13
9
6

REFERENCES

- **FORMAL MODELING WITH PROPOSITIONAL LOGIC**
  Assaf Kfoury

THANK YOU

Joan Orellana Rios