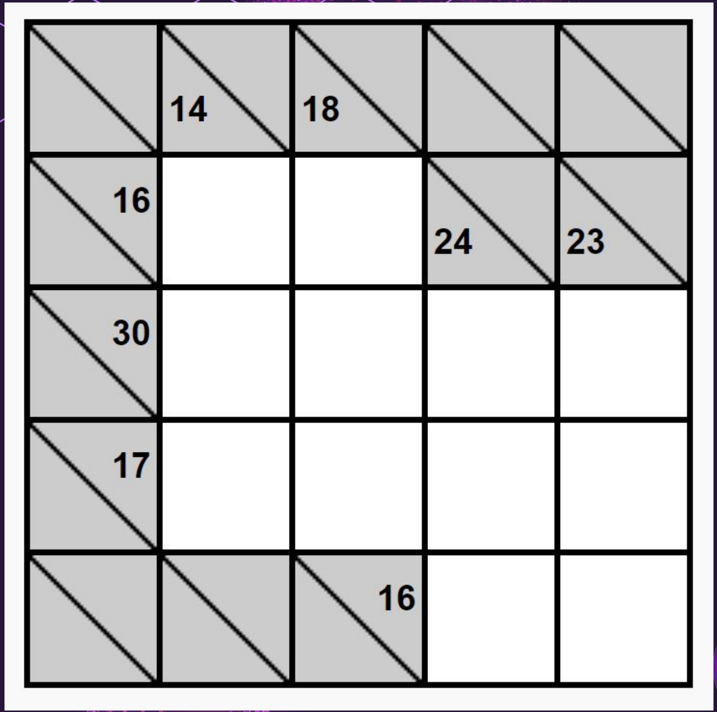
AI Kakuro Puzzle Solver

Intro to AI Mini Project

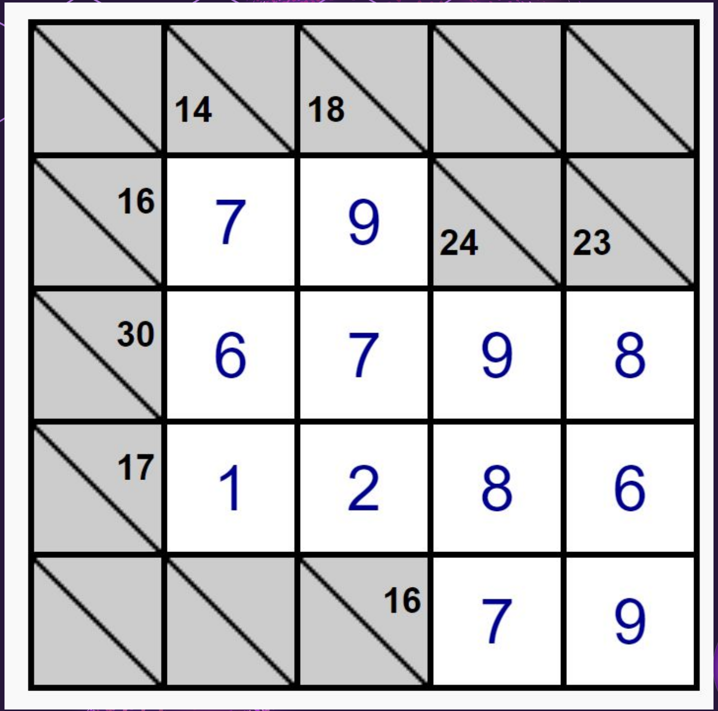
Austin Burcham, Carlos Urbina, Andrew Neeser, Benny Weng

TABLE OF CONTENTS



Kakuro

Kakuro is a grid like puzzle similar to sudoku, where you have to fill in cells where each row and column has no repeat numbers. The extra challenge is to make the sum of each row and column equal to the header number.



Kakuro

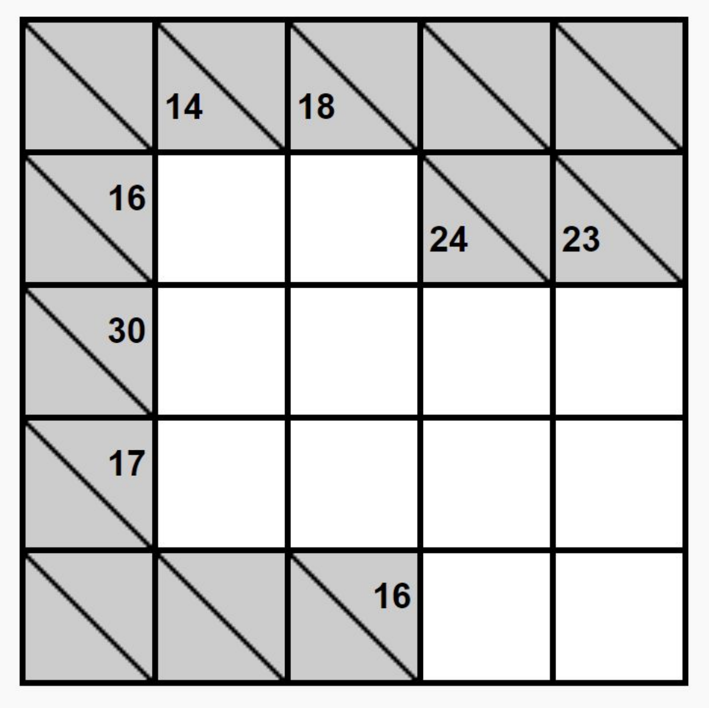
Kakuro is a grid like puzzle similar to sudoku, where you have to fill in cells where each row and column has no repeat numbers. The extra challenge is to make the sum of each row and column equal to the header number.

01

PROBLEM STATEMENT

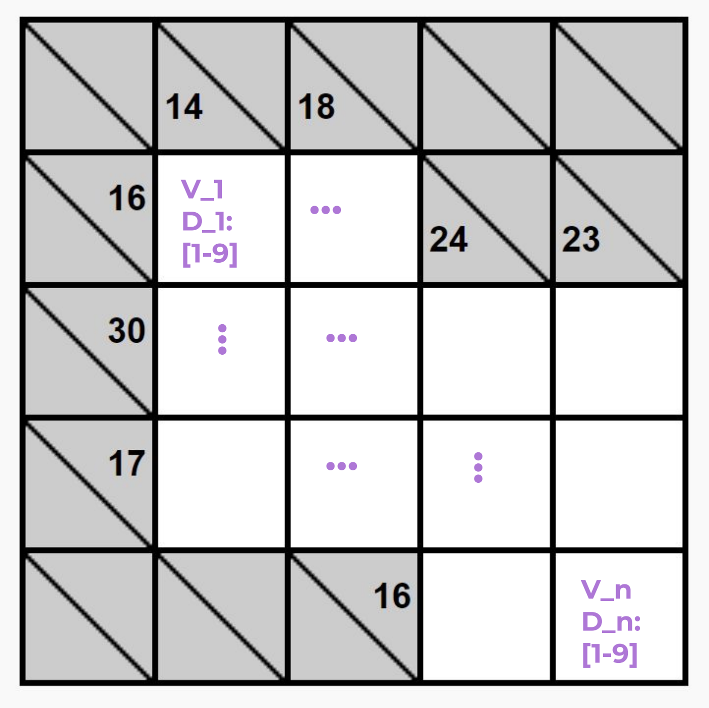
Application to CSP

- Well Suited for CSP:
  - Built-In Constraint Rules
  - Built-In Limited Domains
  - Defined Grid Structure
  - Logical Consistency



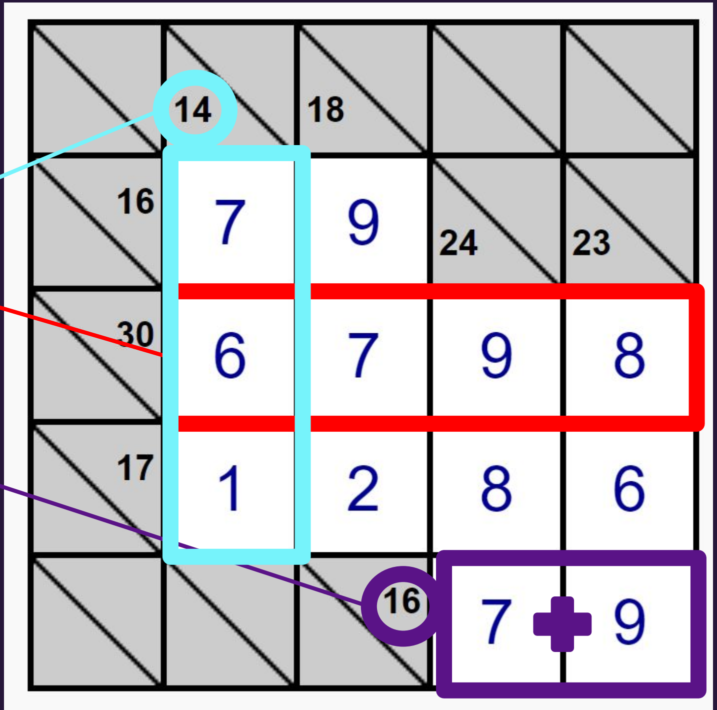
Application to CSP

- Each empty cell is variable
  - (v_1... v_n)
  - Discrete domains (D_1 ... D_n)
    - 1 - 9



Application to CSP

- Three ternary constraint types:
  - Alldiff
    - Rows + Cols
  - Rowsum
  - ColSum



02

Use-Case Scenarios

Educational Tool

- Can be used as an educational tool to help users learn solving techniques
  - Interface will show step-by-step what is going on in each iteration, allowing users to follow along and analyze solving strategies

```
Current Board State:
    17  18  26  11
14 [X][5][9][X]
14 [ ][7][ ][ ]
28 [ ][ ][ ][ ]
16 [X][ ][ ][X]

Current Domains:
    17  18  26  11
14 [X][6789][][X]
14 [123457][789][123457][123457]
28 [12356789][][1235678][12356789]
16 [X][][][X]

Value of 7 selected for cell at row 1 and column 1
```

```
Board state after selection:
    17  18  26  11
14 [X][5][9][X]
14 [ ][7][ ][ ]
28 [ ][ ][ ][ ]
16 [X][ ][ ][X]

Current Board State:
    17  18  26  11
14 [X][5][9][X]
14 [ ][7][ ][ ]
28 [ ][1][ ][ ]
16 [X][ ][ ][X]

Current Domains:
    17  18  26  11
14 [X][6789][][X]
14 [123456][89][123456][123456]
28 [123456789][1234][12345678][123456789]
16 [X][1234][12345678][X]

Value of 1 selected for cell at row 2 and column 1
```

Puzzle Solver

- The application can also be used as a Kakuro board solver allowing users to either find which step they made a mistake on, or whether their solution was correct
  - Will display the correct results at the end and say whether constraints were met or not

```
Current Board State:
    17  18  26  11
14 [X][5][9][X]
14 [ ][7][ ][ ]
28 [ ][ ][ ][ ]
16 [X][ ][ ][X]

Current Domains:
    17  18  26  11
14 [X][6789][][X]
14 [123457][789][123457][123457]
28 [12356789][][1235678][12356789]
16 [X][][][X]

Value of 7 selected for cell at row 1 and column 1
```

```
Solved! Board after backtracking algo executed:
    17  18  26  11
14 [X][6][8][X]
14 [8][1][2][3]
28 [9][4][7][8]
16 [X][7][9][X]
T/F: The board meets constraints? True
```

03

AI Algorithm

1. Recursive Backtracking Algorithm

2. Arc Consistency Inference

3. Check for Completeness or Failure

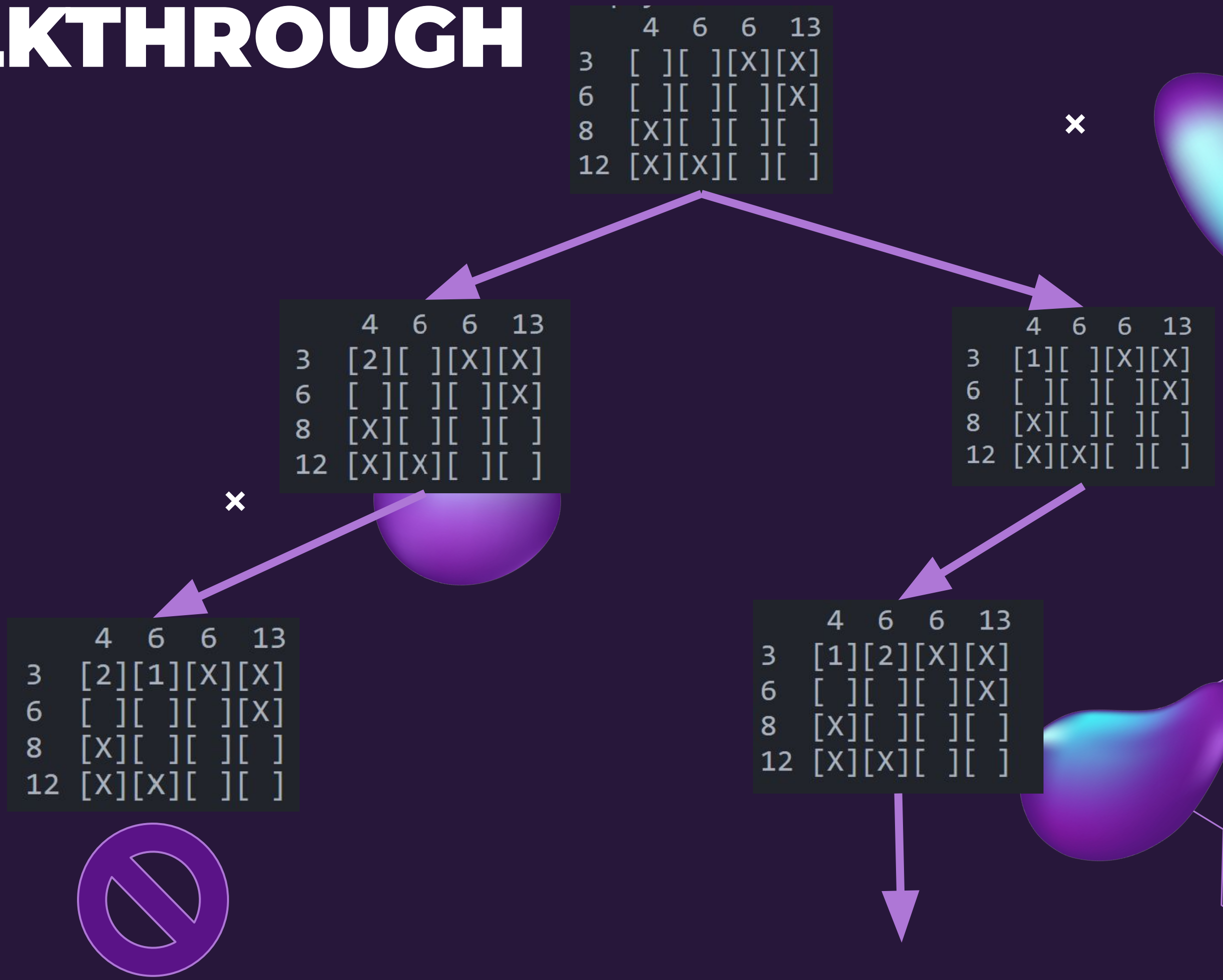
BACKTRACKING ALGORITHM

We approached the puzzle with a backtracking search approach, trying values for cells until failure or completion.

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

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

WALKTHROUGH



Minimum Remaining Values Heuristic

When deciding which cell to assign next, we used the MRV heuristic. This helps us to find failures sooner.

var ← SELECT-UNASSIGNED-VARIABLE(*csp*, *assignment*)

Minimum Remaining Values

- **Minimum Remaining Values** (MRV) heuristic: Choose the **variable** with the **fewest legal left values** in its domain
  - "Most constrained variable" heuristic
  - "Fail-first" heuristic

ARC-CONSISTENCY 3

add $\{var = value\}$ to *assignment*
inferences $\leftarrow$ INFERENCE(*csp*, *var*, *assignment*)

Before the start of backtracking and each time assigning a value, infer new domains for neighboring cells.

- *Alldiff* constraint for each row/column
- Calculate maximum possible value for neighbors
- Calculate only possible value if only one remaining cell in row/column
- Remove value from cell's own domain (for backtracking purposes)

Check for Completeness or Failure

```
Board state after selection:
    4  6  6  13
3  [2][1][X][X]
6  [ ][ ][ ][X]
8  [X][ ][ ][ ]
12 [X][X][ ][ ]

Backtracking from value 1 at row 0 and column 1

Backtracking from value 2 at row 0 and column 0

Current Board State:
    4  6  6  13
3  [1][ ][X][X]
6  [ ][ ][ ][X]
8  [X][ ][ ][ ]
12 [X][X][ ][ ]
```

Backtrack

When any one domain becomes empty, we must backtrack and try new values.

```
    17  18  26  11
14 [X][ ][ ][X]
14 [ ][ ][ ][ ]
29 [ ][ ][ ][ ]
16 [X][ ][ ][X]
T/F: The board meets constraints? False
```

Failure

When a cell's domain becomes empty without being able to backtrack any further, there is no solution.

```
Solved! Board after backtracking algo executed:
    17  18  26  11
14 [X][6][8][X]
14 [8][1][2][3]
28 [9][4][7][8]
16 [X][7][9][X]
T/F: The board meets constraints? True
```

Complete

When all values are assigned and are within constraints, we are done.

04

RESULTS AND DEMONSTRATION

05

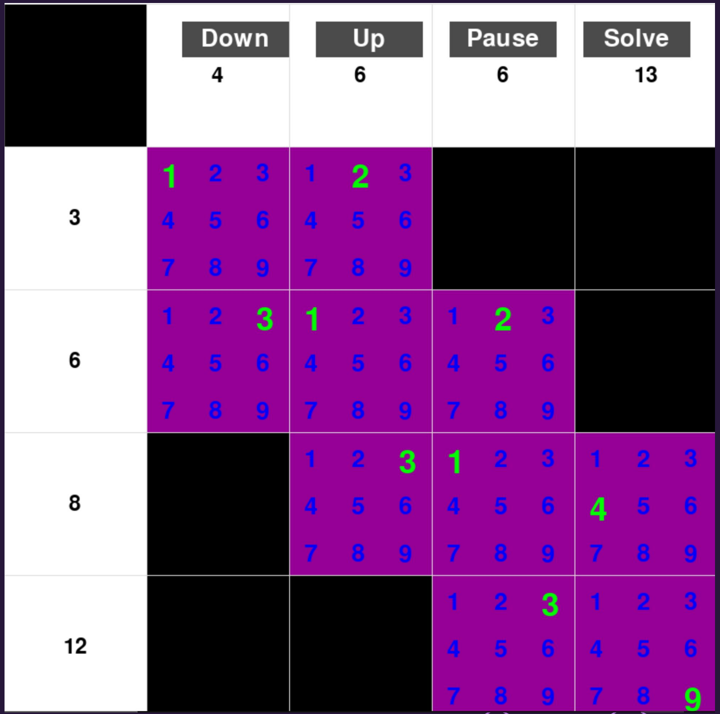
LESSONS LEARNED

Successes

The constraints involved with Kakuro puzzles made it well suited for CSPs. This made writing our backtracking and AC-3 algorithms as straightforward as possible.

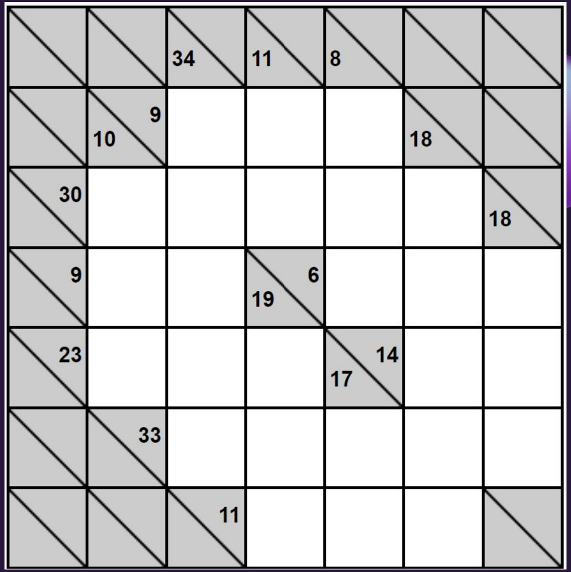
We all learned a lot about how to implement AI solutions for a real system.

As we made progress, we found ways to include more ideas about CSPs from the course such as using heuristics, and using inference.



Limitations of our Implementation

Efficiency can be improved through better domain ordering. A heuristic such as max-conflicts can help to find failures quicker.



Complex puzzles don’t work with our current row and column model, but it would be possible in the future

Q&A

DO YOU HAVE ANY QUESTIONS?