Games, Constraint Satisfaction

CSCI 4511/6511

Joe Goldfrank

Announcements

- Homework 1 is due *today* at 11:55 PM
  - No more than three grace days
- Homework 2 is due on 21 February at 11:55 PM
- Autograder

Algorithms for Games

Adversity

So far:

- The world does not care about us
- This is a simplifying assumption!

Reality:

- The world does not care about us
- ...but it wants things for “itself”
- ...and we don’t want the same things

The Adversary

One extreme:

- Single adversary
    - Adversary wants the *exact opposite* from us
    - If adversary “wins,” we lose

😐

Other extreme:

- An entire world of agents with different values
    - They might want some things similar to us
- “Economics”

😐

Simple Games

- Two-player
- Turn-taking
- Discrete-state
- Fully-observable
- Zero-sum
  - This does some work for us!

We Played A Game

- Pick a partner
- Place 11 pieces of candy between you
- Alternating turns, either:
    - Take one piece
    - Take two pieces
- Last person to take a piece wins all of the candy

Max and Min

- Two players want the opposite of each other
- State takes into account both agents
  - Actions depend on whose turn it is

Minimax

- Initial state s_0
- $\textsc{Actions}(s)$ and $\textsc{To-move}(s)$
- $\textsc{Result}(s, a)$
- $\textsc{Is-Terminal}(s)$
- $\textsc{Utility}(s, p)$

Minimax

Minimax

Algorithm Minimax Search

```
function MINIMAX-SEARCH(game, state)
    player ← game.TO-MOVE(state)
    value, move ← MAX-VALUE(game, state)
    return move

function MAX-VALUE(game, state)
    if game.IS-TERMINAL(state) then
        return game.UTILITY(state, player),null
    v ← −∞
    for each a in game.ACTIONS(state) do
        v2, a2 ←MIN-VALUE(game, game.RESULT(state, a))
        if v2 > v then
            v, move ← v2, a
    return v, move

function MIN-VALUE(game, state)
    if game.IS-TERMINAL(state) then
        return game.UTILITY(state, player),null
    v ← ∞
    for each a in game.ACTIONS(state) do
        v2, a2 ←MAX-VALUE(game, game.RESULT(state, a))
        if v2 < v then
            v, move ← v2, a
    return v, move
```

More Than Two Players

- Two players, two values: v_A, v_B
  - Zero-sum: $v_A = -v_B$
  - Only one value needs to be explicitly represented
- > 2 players:
  - $v_A, v_B, v_C \ldots$
  - Value scalar becomes $\vec{v}$

Society

- >2 players, only one can win
- Cooperation can be rational!

Example:

- A & B: 30% win probability each
- C: 40% win probability
- A & B cooperate to eliminate C
    - $\rightarrow$ A & B: 50% win probability each

...what about friendship?

Minimax Efficiency

Pruning removes the need to explore the full tree.

- Max and Min nodes alternate
- Once *one* value has been found, we can eliminate parts of search
  - Lower values, for Max
  - Higher values, for Min
- Remember highest value (α) for Max
- Remember lowest value (β) for Min

Pruning

Heuristics 😌

- In practice, trees are far too deep to completely search
- Heuristic: replace utility with evaluation function
  - Better than losing, worse than winning
  - Represents chance of winning
- Chance? 🎲🎲
  - Even in deterministic games
  - Why?

More Pruning

- Don’t bother further searching bad moves
  - Examples?
- Beam search
  - Lee Sedol’s singular win against AlphaGo

Other Techniques

- Move ordering
  - How do we decide?
- Lookup tables
  - For subsets of games

Monte Carlo Tree Search

- Many games are too large even for an efficient α-β search 😔
  - We can still play them
- *Simulate* plays of entire games from starting state
  - Update win probability from each node (for each player) based on result
- "Explore/exploit" paradigm for move selection

Choosing Moves

- We want our search to pick good moves
- We want our search to pick unknown moves
- We *don't* want our search to pick bad moves
  - (Assuming they're actually bad moves)

Select moves based on a heuristic.

Games of Luck

- Real-world problems are rarely deterministic
- Non-deterministic state evolution:
  - Roll a die to determine next position
  - Toss a coin to determine who picks candy first
  - Precise trajectory of kicked football[1]
  - Others?

1. Any definition of “football”

Solving Non-Deterministic Games

Previously: Max and Min alternate turns

Now:

- Max
- Chance
- Min
- Chance

😣

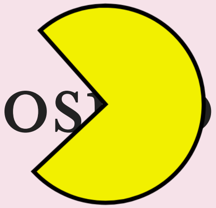
We Played Another Game

- Place 11 pieces of candy between you
- Alternating turns:
  - Choose to take one or two pieces
- *Except:*
  - After choosing, flip two coins, record *total* number of heads[1]
  - If total is divisible by 3, take one less piece than you chose
  - If total is divisible by 5, take one more piece than you chose
  - If total divisible by 15, take no candy
- Last person to take a piece wins all of the candy

1. Keep a running total through the game

Expectiminimax

- "Expected value" of next position



- How does this impact branching factor of the search?

Filled With Uncertainty

What is to be done?

- Pruning is still possible
  - How?
- Heuristic evaluation functions
  - Choose carefully!

Non-Optimal Adversaries

- Is deterministic “best” behavior optimal?
- Are all adversaries rational?

- Expectimax

CSPs

Factored Representation

- Encode relationships between variables and states
- Solve problems with *general* search algorithms
  - Heuristics do not require expert knowledge of problem
  - Encoding problem requires expert knowledge of problem[1]

Why?

1. But it always does.

Constraint Satisfaction

- Express problem in terms of state variables
  - Constrain state variables
- Begin with all variables unassigned
- Progressively assign values to variables
- Assignment of values to state variables that "works:" *solution*

More Formally

- State variables: $X_1, X_2, \ldots, X_n$
- State variable domains: $D_1, D_2, \ldots, D_n$
  - The domain specifies which values are permitted for the state variable
  - Domain: set of allowable variables (or permissible range for continuous variables)[1]
  - Some constraints $C_1, C_2, \ldots, C_m$ restrict allowable values

1. Or a hybrid, such as a union of ranges of continuous variables.

Constraint Types

- Unary: restrict single variable
  - Can be rolled into domain
  - Why even have them?
- Binary: restricts two variables
- Global: restrict "all" variables

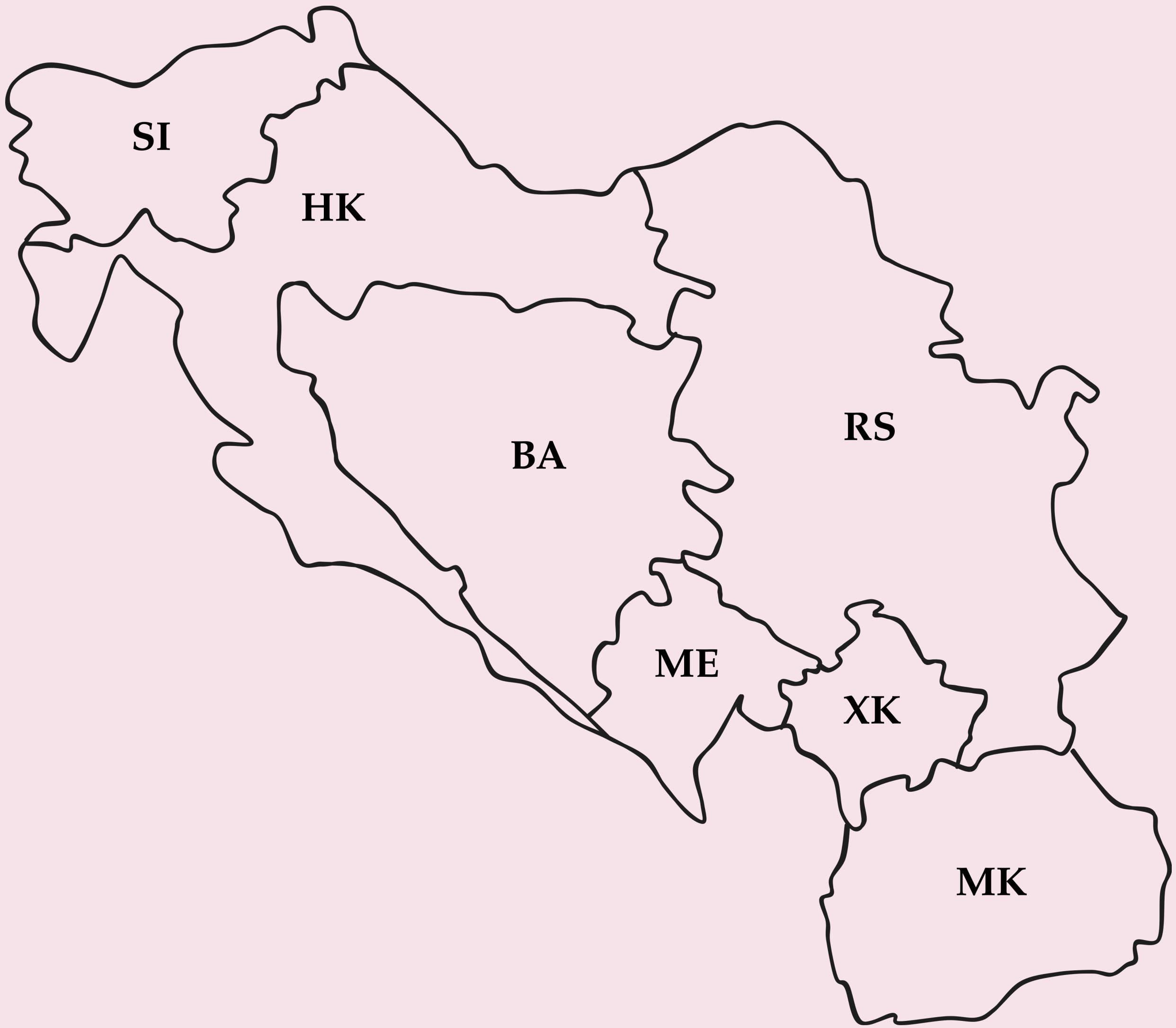
Constraint Examples

- X_1 and X_2 both have real domains, i.e. $X_1, X_2 \in \mathbb{R}$
  - A constraint could be $X_1 < X_2$
- X_1 could have domain $\{\text{red}, \text{green}, \text{blue}\}$ and X_2 could have domain $\{\text{green}, \text{blue}, \text{orange}\}$
  - A constraint could be $X_1 \neq X_2$
- $X_1, X_2, \ldots, X_100 \in \mathbb{R}$
  - Constraint: exactly four of X_i equal 12
  - Rewrite as binary constraint?

Assignments

- Assignments must be to values in each variable's domain
- Assignment violates constraints?
  - Consistency
- All variables assigned?
  - Complete

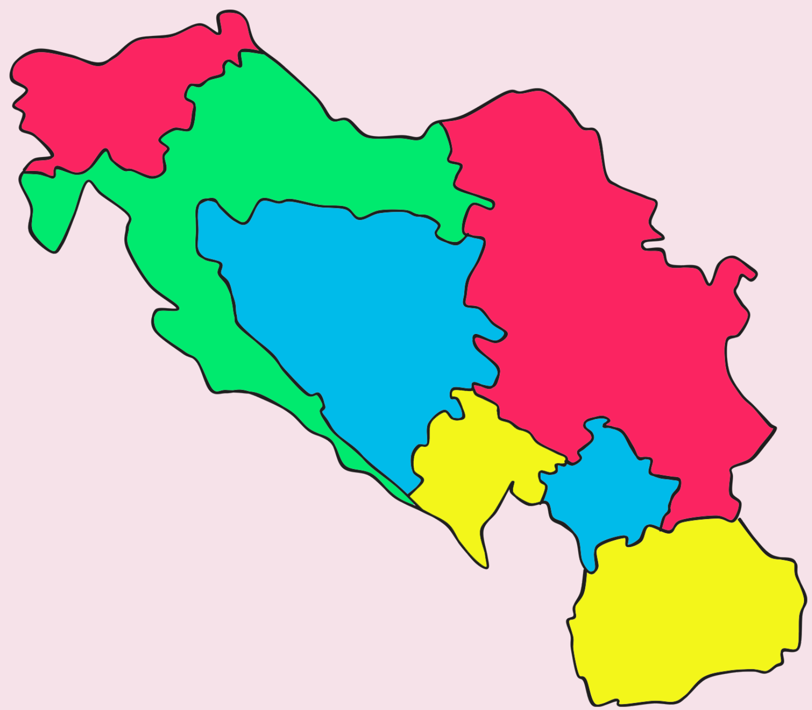
Yugoslavia[1]



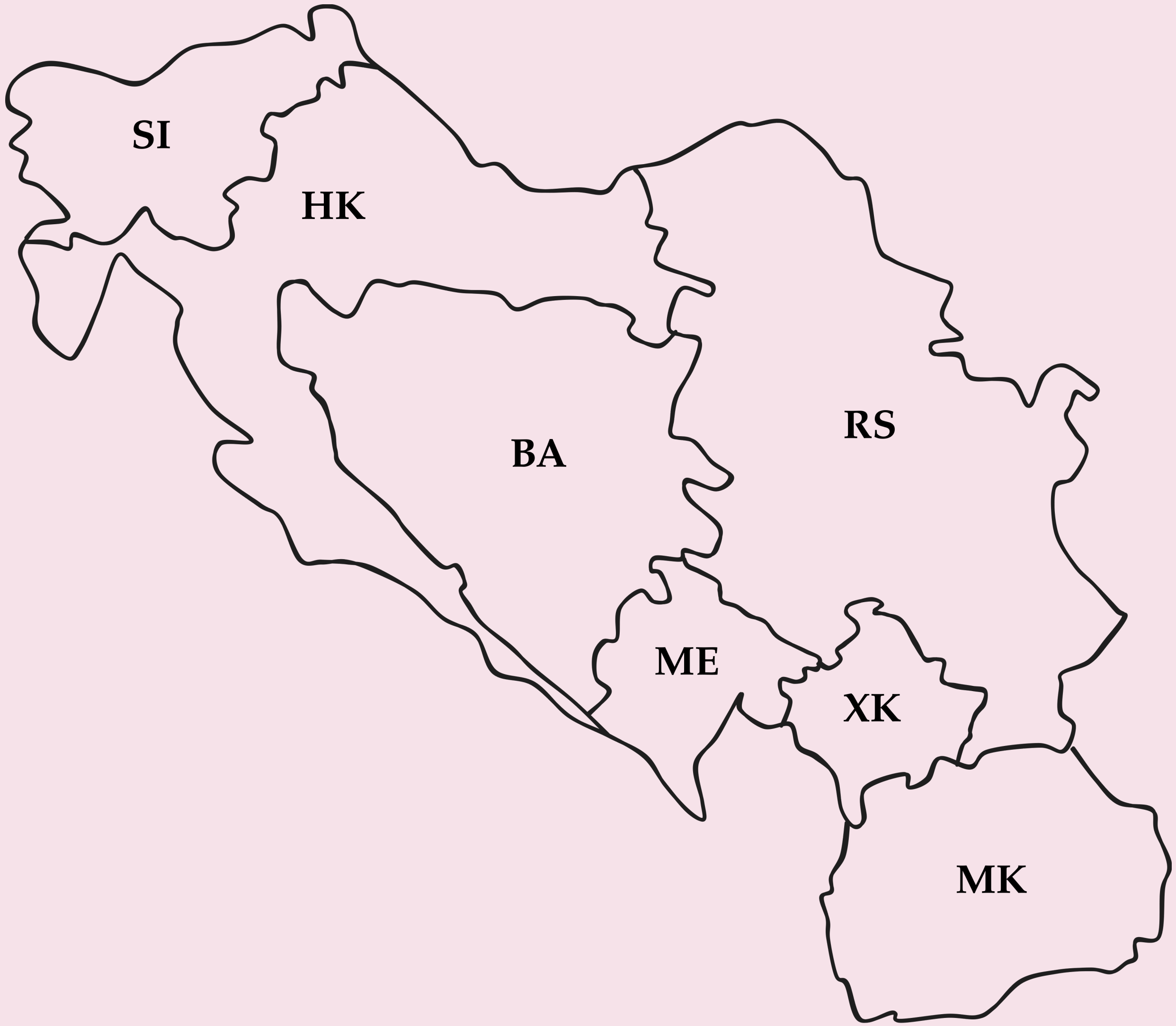
1. One of the most difficult problems of the 20th century

Four-Colorings

Two possibilities:



Formulate as CSP?



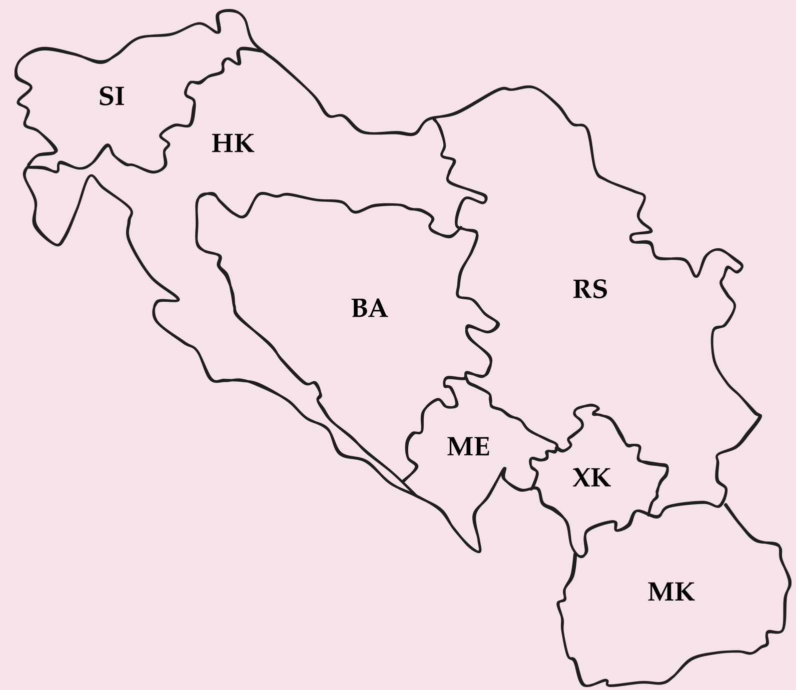
Graph Representations

- Constraint graph:
  - Nodes are variables
  - Edges are constraints
- Constraint hypergraph:
  - Variables are nodes
  - Constraints are nodes
  - Edges show relationship

Why have two different representations?

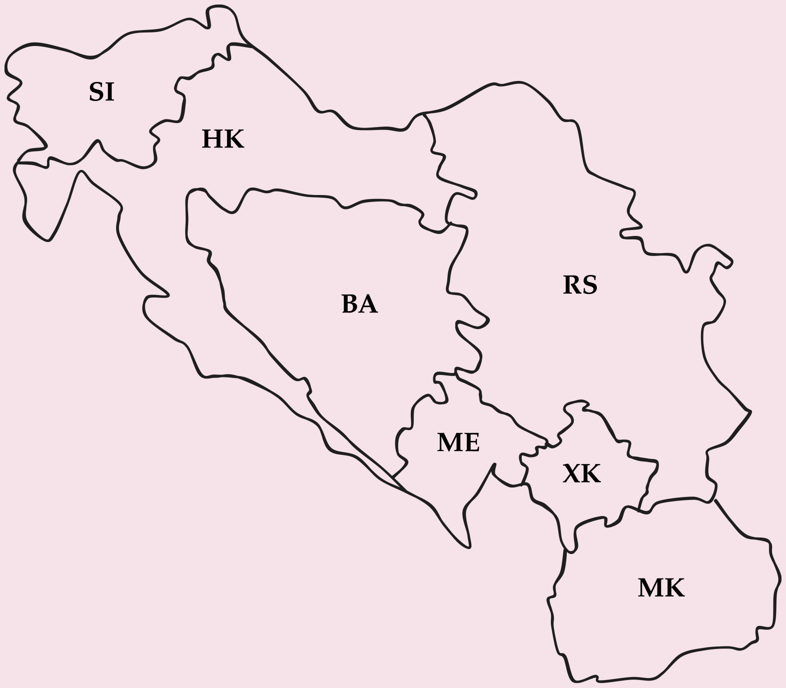
Graph Representation I

Constraint graph: edges are constraints



Graph Representation II

Constraint hypergraph: constraints are nodes



How To Solve It

- We can search!
  - ...the space of consistent assignments
- Complexity $O(d^n)$
  - Domain size d, number of nodes n
- Tree search for node assignment
  - Inference to reduce domain size
- Recursive search

How To Solve It

Algorithm Backtracking Search

---

function $\textsc{Backtracking-Search}(CSP)$
    return $\textsc{Backtrack}(CSP, \{\})$

function $\textsc{Backtrack}(CSP, assignment)$
    if $assignment$ is complete **then**
        return $assignment$
    $var \leftarrow \textsc{Select-Unassigned-Variable}(CSP, assignment)$
    for each $value$ **in** $\textsc{Order-Domain-Variables}(CSP, var, assignment)$ **do**
        if $value$ is consistent with $assignment$ **then**
            $assignment.\textsc{Add}(var = value)$
            $inferences \leftarrow \textsc{Inference}(CSP, var, assignment)$
            if $inferences \neq failure$ **then**
                $CSP.\textsc{Add}(inferences)$
                $result \leftarrow \textsc{Backtrack}(CSP, assignment)$
                if $result \neq failure$ **then**
                    return $result$
                $CSP.\textsc{Remove}(inferences)$
            $assignment.\textsc{Remove}(var = value)$

What Even Is Inference

- Constraints on one variable restrict others:
  - $X_1 \in \{A, B, C, D\}$ and $X_2 \in \{A\}$
  - $X_1 \neq X_2$
  - Inference: $X_1 \in \{B, C, D\}$
- If an unassigned variable has no domain...
  - Failure

Inference

- Arc consistency
  - Reduce domains for pairs of variables
- Path consistency
  - Assignment to two variables
  - Reduce domain of third variable

AC-3

Algorithm AC-3

```
function AC-3(CSP)
    queue ← all arcs in CSP
    while queue is not empty do
        (X_i, X_j) ← Pop(queue)
        if Revise(CSP, X_i, X_j) then
            for each X_k in X_i.Neighbors − {X_j} do
                queue.Add((X_i, X_j))
    return True

function Revise(CSP, X_i, X_j)
    revised ← False
    for each x in D_i do
        if C(X_i = x, X_j) not satisfied for any value in D_j then
            D_i.Remove(x)
            revised ← True
    return revised
```

How To Solve It (Again)

Backtracking search:

- Similar to DFS
- Variables are *ordered*
  - Why?
- Constraints checked each step
- Constraints optionally *propagated*

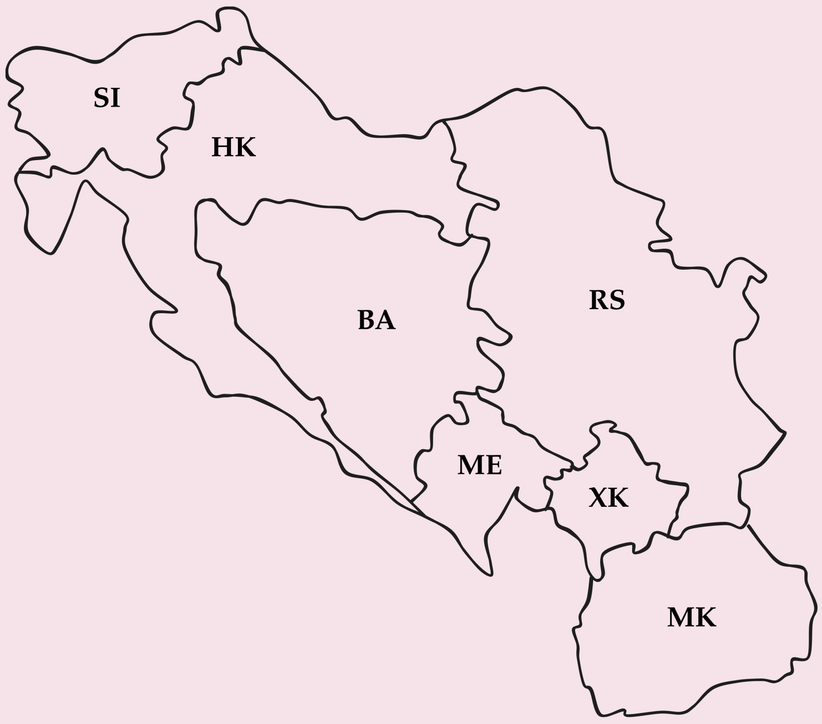
How To Solve It (Again)

Algorithm Backtracking Search

```
function BACKTRACKING-SEARCH(CSP)
    return BACKTRACK(CSP, {})

function BACKTRACK(CSP, assignment)
    if assignment is complete then
        return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(CSP, assignment)
    for each value in ORDER-DOMAIN-VARIABLES(CSP, var, assignment) do
        if value is consistent with assignment then
            assignment.ADD(var = value)
            inferences ← INFERENCE(CSP, var, assignment)
            if inferences ≠ failure then
                CSP.ADD(inferences)
                result ← BACKTRACK(CSP, assignment)
                if result ≠ failure then
                    return result
                CSP.REMOVE(inferences)
            assignment.REMOVE(var = value)
```

Yugoslav Arc Consistency



Ordering

- SELECT-UNASSGINED-VARIABLE($CSP, assignment$)
  - Choose most-constrained variable[1]
- ORDER-DOMAIN-VARIABLES($CSP, var, assignment$)
  - Least-constraining value
- Why?

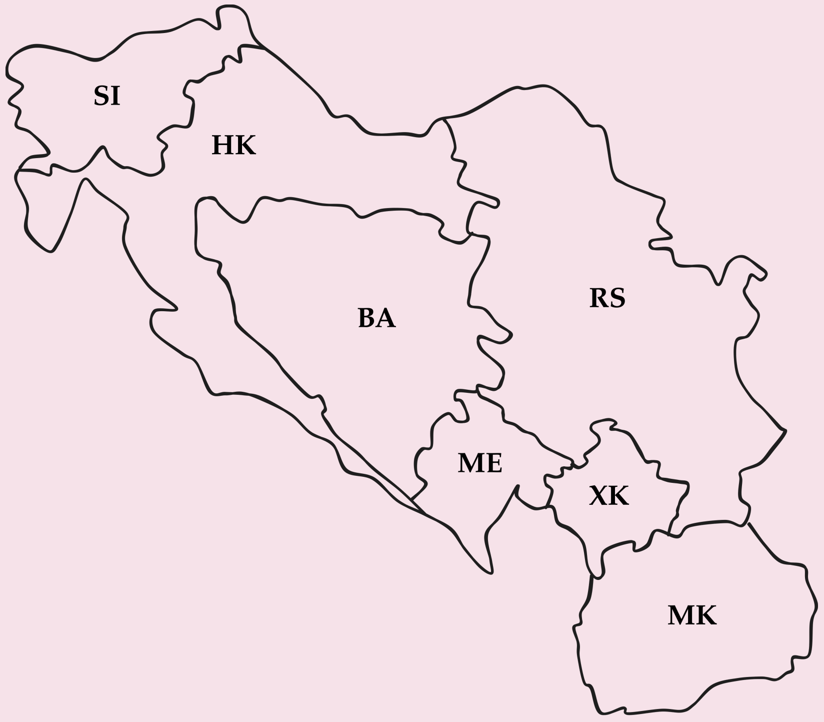
1. or MRV: “Minimum Remaining Values”

Restructuring

Tree-structured CSPs:

- *Linear time* solution
- Directional arc consistency: $X_i \to X_{i+1}$
- Cutsets
- Sub-problems

Cutset Example



(Heuristic) Local Search

- Hill climbing
  - Random restarts
- Simulated annealing
- Fast?
- Complete?
- Optimal?

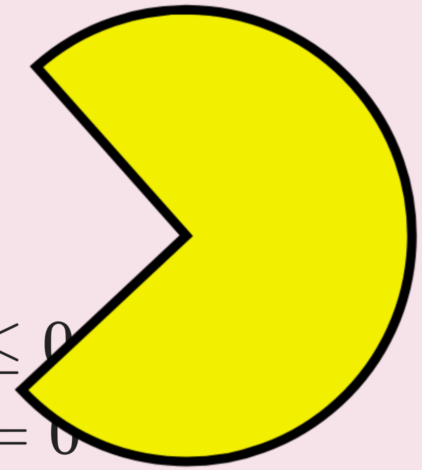
Continuous Domains

- Linear:

$$
\begin{aligned}
\max_{x} \quad & \boldsymbol{c}^T \boldsymbol{x} \\
\text{s.t.} \quad & A\boldsymbol{x} \leq \boldsymbol{b} \\
& \boldsymbol{x} \geq 0
\end{aligned}
$$

- Convex

$$
\begin{aligned}
\min_{x} \quad & f(\boldsymbol{x}) \\
\text{s.t.} \quad & g_i(\boldsymbol{x}) \leq 0 \\
& h_i(\boldsymbol{x}) = 0
\end{aligned}
$$



Is This Even Relevant in 2025?

- Absolutely yes.
- LLMs are bad at CSPs
- CSPs are common in the real world
    - Scheduling
    - Optimization
    - Dependency solvers

Logic Preview

$$R_{HK} \Rightarrow \neg R_{SI}$$
$$G_{HK} \Rightarrow \neg G_{SI}$$
$$B_{HK} \Rightarrow \neg B_{SI}$$
$$R_{HK} \lor G_{HK} \lor B_{HK}$$

...

Goal: find assignment of variables that satisifies conditions

References

- Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 4th Edition, 2020.
- Mykal Kochenderfer, Tim Wheeler, and Kyle Wray. *Algorithms for Decision Making*. 1st Edition, 2022.
- Stanford CS231
- Stanford CS228
- UC Berkeley CS188