Constraint Satisfaction Problems

Stephen G. Ware

CSCI 4525 / 5525

Narrative Intelligence LAB

THE UNIVERSITY of NEW ORLEANS

Classical AI

Classical Artificial Intelligence:

- Knowledge Representation
- Search

Knowledge Representation

So far, we have discussed general search techniques that treat the state as a black box. All we need is:

- A way to transition from one state to another
- A way to know if a state is a goal
- A way to estimate distance to the goal (optional)

Knowledge Representation

By considering the internal structure of the knowledge representation, we can develop search techniques which take advantage of its properties.

Challenge: Designing KR's which are general enough to be used for a wide variety of things but specific enough that we can take advantage of their structure.

Constraint Satisfaction Problems

- A set of **variables**, which can be assigned values
- Each variable has a **domain** of legal values
- **Constraints** describing which values are legal

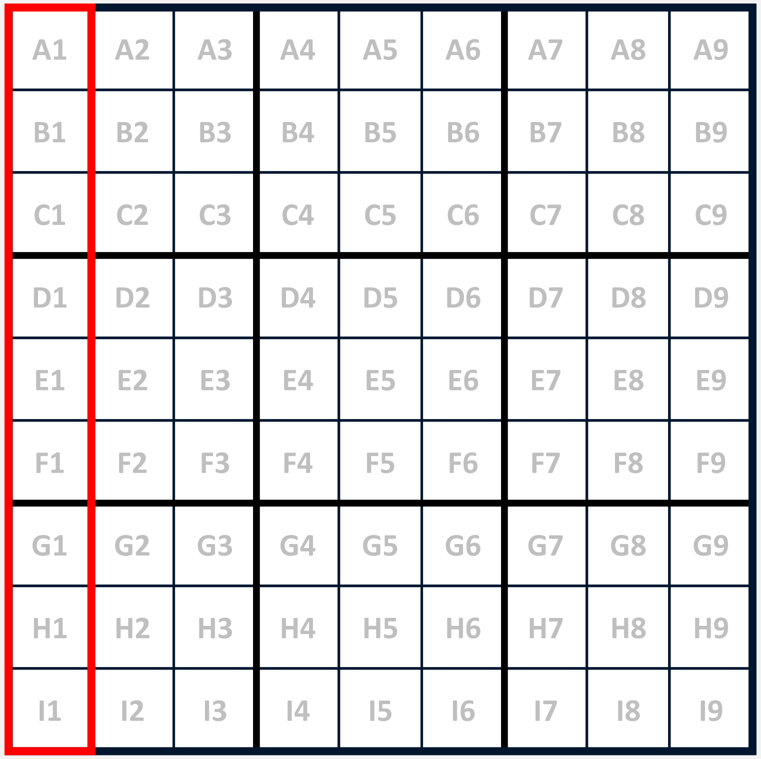
Goal: Find an assignment of values to variables which satisfies all constraints.

Sudoku Puzzle

A1	A2	A3	A4	A5	A6	A7	A8	A9
B1	B2	B3	B4	B5	B6	B7	B8	B9
C1	C2	C3	C4	C5	C6	C7	C8	C9
D1	D2	D3	D4	D5	D6	D7	D8	D9
E1	E2	E3	E4	E5	E6	E7	E8	E9
F1	F2	F3	F4	F5	F6	F7	F8	F9
G1	G2	G3	G4	G5	G6	G7	G8	G9
H1	H2	H3	H4	H5	H6	H7	H8	H9
I1	I2	I3	I4	I5	I6	I7	I8	I9

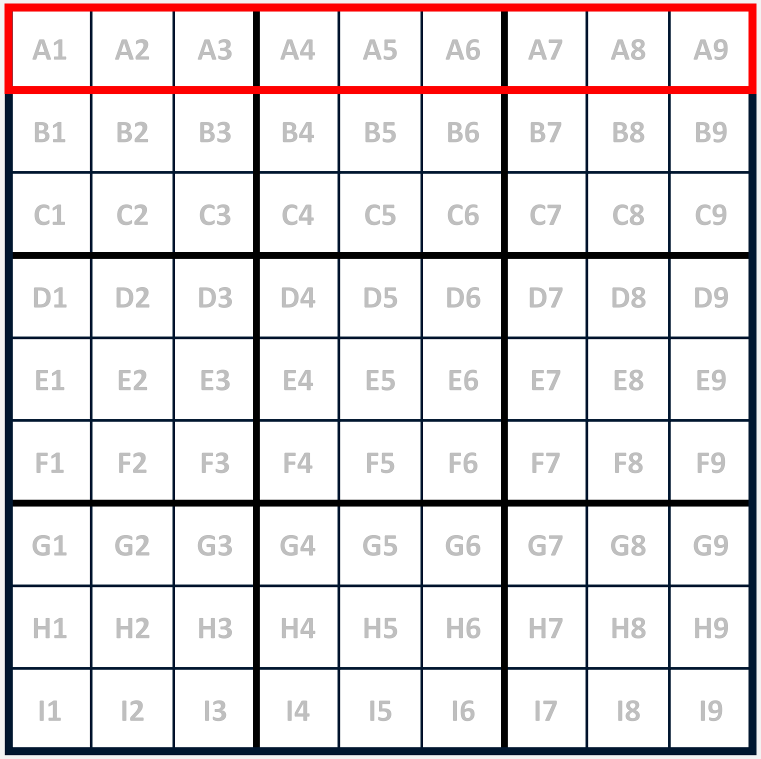
- Each unit holds a single digit {1,2,3,4,5,6,7,8,9}

Sudoku Puzzle


A1 A2 A3 A4 A5 A6 A7 A8 A9
B1 B2 B3 B4 B5 B6 B7 B8 B9
C1 C2 C3 C4 C5 C6 C7 C8 C9
D1 D2 D3 D4 D5 D6 D7 D8 D9
E1 E2 E3 E4 E5 E6 E7 E8 E9
F1 F2 F3 F4 F5 F6 F7 F8 F9
G1 G2 G3 G4 G5 G6 G7 G8 G9
H1 H2 H3 H4 H5 H6 H7 H8 H9
I1 I2 I3 I4 I5 I6 I7 I8 I9

- Each unit holds a single digit {1,2,3,4,5,6,7,8,9}
- Every unit in a column must be unique

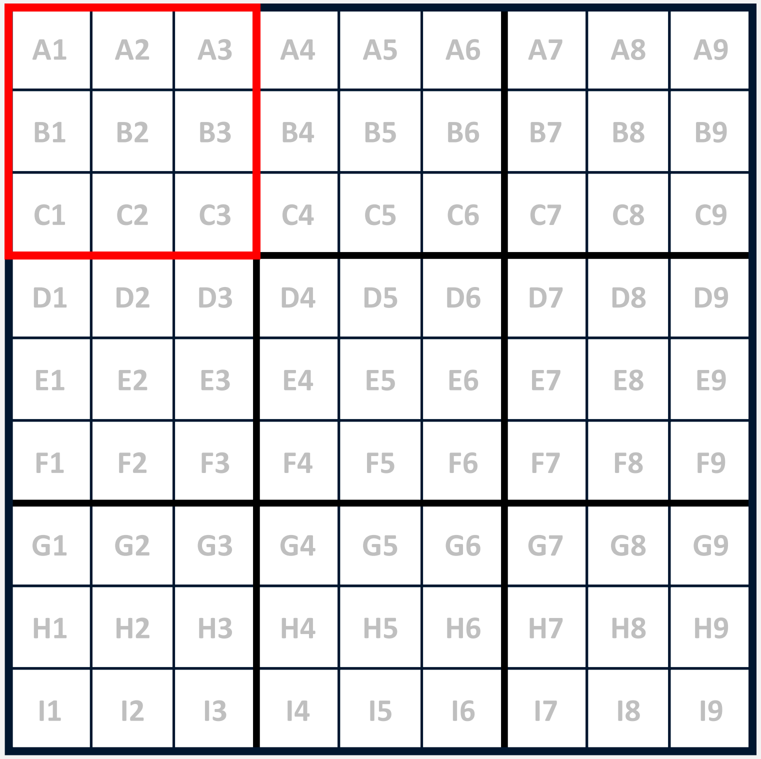
Sudoku Puzzle



A1	A2	A3	A4	A5	A6	A7	A8	A9
B1	B2	B3	B4	B5	B6	B7	B8	B9
C1	C2	C3	C4	C5	C6	C7	C8	C9
D1	D2	D3	D4	D5	D6	D7	D8	D9
E1	E2	E3	E4	E5	E6	E7	E8	E9
F1	F2	F3	F4	F5	F6	F7	F8	F9
G1	G2	G3	G4	G5	G6	G7	G8	G9
H1	H2	H3	H4	H5	H6	H7	H8	H9
I1	I2	I3	I4	I5	I6	I7	I8	I9

- Each unit holds a single digit {1,2,3,4,5,6,7,8,9}
- Every unit in a column must be unique
- Every unit in a row must be unique

Sudoku Puzzle



A1	A2	A3	A4	A5	A6	A7	A8	A9
B1	B2	B3	B4	B5	B6	B7	B8	B9
C1	C2	C3	C4	C5	C6	C7	C8	C9
D1	D2	D3	D4	D5	D6	D7	D8	D9
E1	E2	E3	E4	E5	E6	E7	E8	E9
F1	F2	F3	F4	F5	F6	F7	F8	F9
G1	G2	G3	G4	G5	G6	G7	G8	G9
H1	H2	H3	H4	H5	H6	H7	H8	H9
I1	I2	I3	I4	I5	I6	I7	I8	I9

- Each unit holds a single digit {1,2,3,4,5,6,7,8,9}
- Every unit in a column must be unique
- Every unit in a row must be unique
- Every unit in a 3x3 square must be unique

Sudoku as CSP

A1	A2	A3	A4	A5	A6	A7	A8	A9
B1	B2	B3	B4	B5	B6	B7	B8	B9
C1	C2	C3	C4	C5	C6	C7	C8	C9
D1	D2	D3	D4	D5	D6	D7	D8	D9
E1	E2	E3	E4	E5	E6	E7	E8	E9
F1	F2	F3	F4	F5	F6	F7	F8	F9
G1	G2	G3	G4	G5	G6	G7	G8	G9
H1	H2	H3	H4	H5	H6	H7	H8	H9
I1	I2	I3	I4	I5	I6	I7	I8	I9

Variables

A1, A2, A3, … I8, I9

Domain

A1 = {1,2,3,4,5,6,7,8,9}

A2 = {1,2,3,4,5,6,7,8,9} …

Constraints

A1≠A2, A1≠A3, A1≠A4, …

Naïve Sudoku Search

```
While any variable is unassigned:
     Assign a value to an unassigned variable.
```

81 variables, each with 9 possible values.

Branching factor ≈ 81 * 9 = 729

Complexity of search: $O(729^{81})$

Constraint Propagation

The value of phrasing a problem as a CSP is that we can use general, domain-independent knowledge about the internal structure of the state to dramatically reduce the amount of search.

Information about one variable limits the values that other variables can have.

1-Consistency

A **unary constraint** applies to a single variable.

We say that a CSP is **1-consistent** when all of its unary constraints are satisfied.

2-Consistency

A **binary constraint** applies to a pair of variables.

We say that a CSP is **2-consistent** when all of its binary constraints are satisfied.

It is always possible to transform n-ary constraints into a set of binary constraints.

3-Consistency

We say that two variables X and Y are **3-consistent** with respect to variable Z if, for every assignment of values to X and Y that obeys the constraints on X and Y, there is an assignment to Z which obeys all constraints on X, Y, and Z.

We say the whole CSP is 3-consistent when every pair of variables is 3-consistent.

k-Consistency

We say that a CSP is ***k*-consistent** when, for every valid assignment to *k*-1 variables, a consistent value can always be assigned to any *k*th variable.

Computing *k*-consistency becomes exponentially more expensive for higher values of *k*, so in practice, many algorithms compute 2-consistency, but rarely higher than 3-consistency.

Sudoku as CSP

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

unary constraints

(A3=3) (B1=9) (C3=1) …

(A1≠A2 ≠A3 ≠A4 ≠A5 ≠A6 ≠A7 ≠A8 ≠A9) …

9-ary constraint

Sudoku as CSP

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

unary constraints

$(A3=3)$ $(B1=9)$ $(C3=1)$ …

$(A1\neq A2)$ $(A1\neq A3)$ $(A1\neq A4)$
$(A1\neq A5)$ $(A1\neq A6)$ $(A1\neq A7)$…
$(A7\neq A8)$ $(A7\neq A9)$ $(A8\neq A9)$

binary constraints

Sudoku as CSP

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

1-consistency:

A3={1,2,3,4,5,6,7,8,9}

Constraints:

(A3=3) …

Sudoku as CSP

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

1-consistency:

A3={~~1,2,~~3~~,4,5,6,7,8,9~~}

Constraints:

(A3=3) …

Sudoku as CSP

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

2-consistency:

A1={1,2,3,4,5,6,7,8,9}

Constraints:

(A1≠A3) (A1≠A5) (A1≠A7)
(A1≠B1) (A1≠E1) (A1≠H1)
(A1≠C3) …

Sudoku as CSP

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

2-consistency:

A1={1,2,~~3,~~4,5,6,7,8,9}

Constraints:

(A1≠A3) (A1≠A5) (A1≠A7) (A1≠B1) (A1≠E1) (A1≠H1) (A1≠C3) …

Sudoku as CSP

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

2-consistency:

A1={1,~~2,3,~~4,5,6,7,8,9}

Constraints:

(A1≠A3) (A1≠A5) (A1≠A7)
(A1≠B1) (A1≠E1) (A1≠H1)
(A1≠C3) …

Sudoku as CSP

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

2-consistency:

A1={1,~~2,3,~~4,5,~~6,~~7,8,9}

Constraints:

(A1≠A3) (A1≠A5) (A1≠A7)
(A1≠B1) (A1≠E1) (A1≠H1)
(A1≠C3) …

Sudoku as CSP

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

2-consistency:

A1={1,~~2,3,~~4,5,~~6,~~7,8~~,9~~}

Constraints:

$(A1 \neq A3)$ $(A1 \neq A5)$ $(A1 \neq A7)$
$(A1 \neq B1)$ $(A1 \neq E1)$ $(A1 \neq H1)$
$(A1 \neq C3)$ …

Sudoku as CSP

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

2-consistency:

A1={1,~~2,3,~~4,5,~~6,7,~~8~~,9~~}

Constraints:

$(A1 \neq A3)$ $(A1 \neq A5)$ $(A1 \neq A7)$
$(A1 \neq B1)$ $(A1 \neq E1)$ $(A1 \neq H1)$
$(A1 \neq C3)$ …

Sudoku as CSP

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

2-consistency:

A1={1,~~2,3,~~4,5,~~6,7,8,9~~}

Constraints:

(A1≠A3) (A1≠A5) (A1≠A7)
(A1≠B1) (A1≠E1) (A1≠H1)
(A1≠C3) …

Sudoku as CSP

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

2-consistency:

A1={~~1,2,3,~~4,5,~~6,7,8,9~~}

Constraints:

(A1≠A3) (A1≠A5) (A1≠A7)
(A1≠B1) (A1≠E1) (A1≠H1)
(A1≠C3) …

AC-3 Algorithm

```
Let Q be a set of binary constraints.
While Q is not empty:
    Remove a constraint (xi,xj) from Q.
    Revise the domain of xi based on (xi,xj).
    If xi's domain is empty, fail.
    If xi's domain has changed:
        For each constraint like (xk,xi):
            Add (xk,xi) to Q.
Return success.
```

AC-3 Revise

```
To simplify xi based on the constraint (xi,xj):
Let result = false.
For each value i in the xi’s domain:
    For each value j in xj’s domain:
        If (xi=i,xj=j) satisfies the constraint,
            Do not remove i from xi’s domain.
    If no value j was found,
        Remove i from xi’s domain.
        Let result = true.
Return result.
```

AC-3 Example

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

B6={5}

E6={4,5}

I6={4,7}

Remove a constraint from the queue.

Constraint:

Queue: (E6≠B6) …

AC-3 Example

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

B6={5}

E6={4,5}

I6={4,7}

Remove a constraint from the queue.

Constraint: (E6≠B6)

Queue: …

AC-3 Example

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

B6={5}

E6={4,5}

I6={4,7}

Revise E6 based on (E6≠B6)

Constraint: (E6≠B6)

Queue: …

AC-3 Example

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

B6={5}

E6={4,5}

I6={4,7}

Can 4 be removed from E6's domain?

Constraint: (E6≠B6)

Queue: …

AC-3 Example

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

B6={5}

E6={4,5}

I6={4,7}

Can 4 be removed from E6’s domain?

No: (E6=4,B6=5)

Constraint: (E6≠B6)

Queue: …

AC-3 Example

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

B6={5}

E6={4,5}

I6={4,7}

Can 5 be removed from E6's domain?

Constraint: (E6≠B6)

Queue: …

AC-3 Example

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

B6={5}

E6={4,5}

I6={4,7}

Can 5 be removed from E6's domain?
Yes!

Constraint: (E6≠B6)

Queue: …

AC-3 Example

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

B6={5}

E6={4~~,5~~}

I6={4,7}

Can 5 be removed from E6's domain?
Yes!

Constraint: (E6≠B6)

Queue: …

AC-3 Example

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

B6={5}

E6={4~~,5~~}

I6={4,7}

Add constraints on E6 back into the queue.

Constraint:

Queue: …

AC-3 Example

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

B6={5}

E6={4~~,5~~}

I6={4,7}

Add constraints on E6 back into the queue.

Constraint:

Queue: … $(I6 \neq E6)$ …

AC-3 Example

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

B6={5}

E6={4~~,5~~}

I6={4,7}

Remove a constraint from the queue.

Constraint:

Queue: … (I6≠E6) …

AC-3 Example

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

B6={5}

E6={4~~,5~~}

I6={4,7}

Remove a constraint from the queue.

Constraint: (I6≠E6)

Queue: …

AC-3 Example

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

B6={5}

E6={4~~,5~~}

I6={4,7}

Revise I6 based on (I6≠E6)

Constraint: (I6≠E6)

Queue: …

AC-3 Example

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

B6={5}

E6={4~~,5~~}

I6={4,7}

Can 4 be removed From I6's domain?

Constraint: (I6≠E6)

Queue: …

AC-3 Example

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

B6={5}

E6={4~~,5~~}

I6={4,7}

Can 4 be removed From I6's domain?

Yes!

Constraint: (I6≠E6)

Queue: …

AC-3 Example

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

B6={5}

E6={4~~,5~~}

I6={~~4,~~7}

Can 4 be removed
From I6's domain?
Yes!

Constraint: (I6≠E6)

Queue: …

Constraint Propagation

Every variable's domain is reduced to exactly 1 value?

Problem solved!

Some variable's domain is reduced to nothing?

Problem has no solution!

Some domains are reduced?

Search will now be easier.

CSP Search

Many problems cannot be solved simply through constraint propagation; search is required.

Idea: Try assigning one value to a variable. If that does not work, backtrack.

Commutativity

At each iteration of the search, if there are x variables, each with a domain of size y, what is the branching factor?

Naïve solution: xy

CSPs are **commutative**, meaning that the order actions are taken in does not matter.

Better solution: y

CSP Search

```
To solve a CSP:
    If every domain has exactly 1 value, return success.
    If any constraint is violated, return failure.
    Choose a variable v whose domain has more than 1.
    For every value x in v’s domain:
        Remove every value but x from v’s domain.
        If the new CSP can be solved, return success.
    Return failure.
```

Improving CSP Search

Because all CSPs share a common structure, there are several domain-independent improvements we can make to the search process.

All improvements share a common goal: to reduce the branching factor of the search and to get to success or failure as quickly as possible (“fail fast”).

How can we improve this?

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

A1 = {4,5}

A2 = {4,5,6,7,8}

First choice: A1 = 4.

Second choice: A2 = 4.

Constraint violated! Fail.

Interleaving Search & Inference

```
To solve a CSP:
    If every domain has exactly 1 value, return success.
    If any constraint is violated, return failure.
    Choose a variable v whose domain has more than 1.
    For every value x in v’s domain:
        Remove every value but x from v’s domain.
        Propagate constraints.
        If the new CSP can be solved, return success.
    Return failure.
```

Interleaving Search & Inference

```
To solve a CSP:
    If every domain has exactly 1 value, return success.
    ~~If any constraint is violated, return failure.~~
    Choose a variable v whose domain has more than 1.
    For every value x in v’s domain:
        Remove every value but x from v’s domain.
        Propagate constraints.
        If the new CSP can be solved, return success.
    Return failure.
```

Interleaving Search & Inference

```
To solve a CSP:
    If every domain has exactly 1 value, return success.
    If any domain has 0 values, return failure.
    Choose a variable v whose domain has more than 1.
    For every value x in v’s domain:
        Remove every value but x from v’s domain.
        Propagate constraints.
        If the new CSP can be solved, return success.
    Return failure.
```

Interleaving Search & Inference

Method 1: After assigning a value to variable *v*, make *v* 2-consistent.

(i.e. check all binary constraints that involve *v*)

Method 2: Do full constraint propagation, but rather than initializing the queue to be all constraints, initialize it to be only constraints involving *v*.

Interleaving Search & Inference

Method 1 is faster, but may miss some information which can be inferred.

Method 2 is slower, but discovers more information.

Most large problems are so large than they can only be solved via dramatic reductions in the branching factor, so Method 2 is generally preferred.

Which to assign next?

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

A1 = {4,5}

A2 = {4,5,6,7,8}

Variable Ordering

```
To solve a CSP:
    If every domain has exactly 1 value, return success.
    If any domain has 0 values, return failure.
    Choose a variable v whose domain has more than 1.
    For every value x in v’s domain:
        Remove every value but x from v’s domain.
        Propagate constraints.
        If the new CSP can be solved, return success.
    Return failure.
```

Variable Ordering

```
To solve a CSP:
    If every domain has exactly 1 value, return success.
    If any domain has 0 values, return failure.
    **Choose a variable v** whose domain has more than 1.
    For every value x in v’s domain:
        Remove every value but x from v’s domain.
        Propagate constraints.
        If the new CSP can be solved, return success.
    Return failure.
```

Variable Ordering

Goal:

- Minimize the branching factor.
- Find vales which make the current set of variable/value assignments fail as fast as possible, thus pruning the search tree.

Strategy: Choose the "most-constrained variable," i.e. the one with the fewest values in its domain.

Variable Ordering

Goal: Learn as much as we can after assigning a value to a variable.

Strategy: Choose the variable that is involved in the most constraints.

Generally, search is improved more by choosing the variable with the fewest values, but this strategy is helpful for breaking ties (i.e. many variables have equally small domains).

Which value to assign?

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

A1 = {4,5}

A2 = {4,5,6,7,8}

We have decided to choose a value for A2. Which is best?

Value Ordering

```
To solve a CSP:
    If every domain has exactly 1 value, return success.
    If any domain has 0 values, return failure.
    Choose a variable v whose domain has more than 1.
    For every value x in v’s domain:
        Remove every value but x from v’s domain.
        Propagate constraints.
        If the new CSP can be solved, return success.
    Return failure.
```

Value Ordering

```
To solve a CSP:
    If every domain has exactly 1 value, return success.
    If any domain has 0 values, return failure.
    Choose a variable v whose domain has more than 1.
    **For every value x in v’s domain:**
        Remove every value but x from v’s domain.
        Propagate constraints.
        If the new CSP can be solved, return success.
    Return failure.
```

Value Ordering

Goal: Leave ourselves as much flexibility as possible when making future decisions.

Strategy: Choose the “least-constraining value,” i.e. the one which reduces the domains of other variables by as little as possible.

In other words: Choosing a value has consequences on other variables, so try to minimize those consequences.

Improved CSP Search

```
To solve a CSP:
    If every domain has exactly 1 value, return success.
    If any domain has 0 values, return failure.
    Choose the most-constrained variable v.
    For each x in domain, from least to most-constraining:
        Remove every value but x from v’s domain.
        Propagate constraints.
        If the new CSP can be solved, return success.
    Return failure.
```

Can this be avoided?

A1	A2	3	A4	2	A6	6	A8	A9
9	B2	B3	3	B5	5	B7	B8	1
C1	C2	1	8	C5	6	4	C8	C9
D1	D2	8	1	D5	2	9	D8	D9
7	E2	E3	E4	E5	E6	E7	E8	8
F1	F2	6	7	F5	8	2	F8	F9
G1	G2	2	6	G5	9	5	G8	G9
8	H2	H3	2	H5	3	H7	H8	9
I1	I2	5	I4	1	I6	3	I8	I9

A1 = {4,5}

A2 = {4,5,6,7,8}

Choices:

A1 = 5. (wrong)

A2 = 4.

…

Failure! Backtrack to A2.

Backjumping

Goal: Rather than always backtracking to the previous choice, backtrack all the way to the choice which needs to be made differently.

Simple backjumping prunes the same branches of the search tree that constraint propagation would prune, so it is redundant for our improved search.

Other Improvements

- Conflict-Directed Backjumping

  More intelligent, non-redundant backjumping.

- Constraint Learning

  When failure happens, figure out which subset of variable/value pairs caused it to fail, and record these as a "no good" so that other branches of the search know not to try that combination again.

- Local search