CS 486/686
Constraint Satisfaction Problems

Yuntian Deng

Lecture 4

RN 6.1–6.3 · PM 4.1–4.4

Reminders

DUE — **Tue May 26**

Chat 1 (Intro & uninformed search) — due 11:59 PM.

RELEASED TODAY — **Due Thu May 28**

Chat 2 (Heuristic search & CSPs) — two-day extension from the original Tue May 26.

DISTINGUISHED LECTURE — **Tue May 26, 10:00–11:30 AM · DC 1302**

Prof. Kyunghyun Cho (NYU) — co-inventor of *attention* (Bahdanau, Cho, Bengio, ICLR 2015) and the *GRU*. **Highly recommended.**

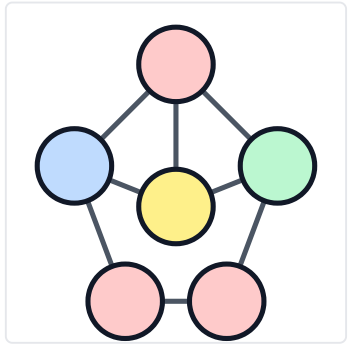
Learning goals

- Formulate a real-world problem as a CSP
- Trace backtracking search
- Verify whether a constraint is arc-consistent
- Trace the AC-3 algorithm
- Trace backtracking *combined* with arc consistency

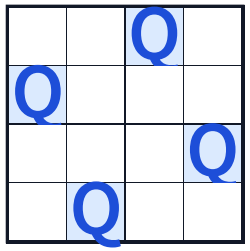
CSPs you've seen

Crosswords

Sudoku



Graph colouring



N-queens

Naive idea: generate-and-test

Enumerate every assignment, test if it satisfies the constraints.

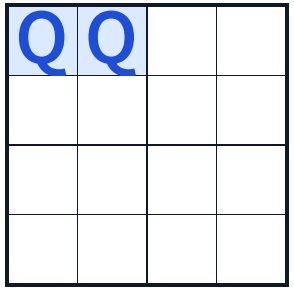
```
A=1, B=1, C=1, D=1, E=1   # fail
A=1, B=1, C=1, D=1, E=2   # fail
…
```

5 variables, domain size 4: $4^5 = 1024$ assignments. Doesn't scale.

And it's wasteful: many constraints can be checked on *partial* assignments.

Search algorithms are structure-blind

A search algorithm sees only "generate successors" and "is this a goal?".



Two queens already in the same row — clearly a dead end — yet the search keeps generating successors. We need to **see the structure**.

A CSP is a triple (X, D, C)

Variables X

$\{X_1, X_2, \ldots, X_n\}$

Domains D

Each X_i draws values from a finite set D_i.

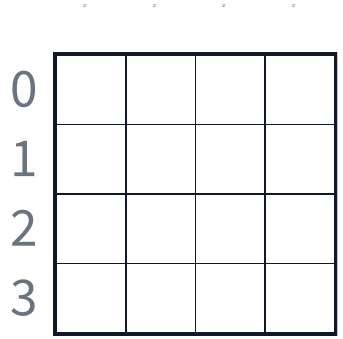
Constraints C

Each $c \in C$ restricts allowed combinations of variable values.

Solution: an assignment to all variables that satisfies every constraint.

4-queens: variables and domains

- One queen per column ⇒ track its *row*.
- **Variables:** x_0, x_1, x_2, x_3 (column index).
- **Domains:** $D_{x_i} = \{0, 1, 2, 3\}$.



4-queens: constraints

No pair of queens in the same row or diagonal.

$$\forall i \neq j: \quad (x_i \neq x_j) \ \wedge \ (|x_i - x_j| \neq |i - j|)$$

Columns 0, 1: $(x_0 \neq x_1) \ \wedge \ (|x_0 - x_1| \neq 1)$.

> No *column* constraint needed — our variables already encode "one queen per column".

Q: which constraints do we need?

Q. Given the variables $x_0, \ldots, x_3$ and their domains, which row/column/diagonal constraints must we enforce?

A. No two queens in the same row

B. No two queens in the same column

C. No two queens on the same diagonal

D. Two of (A), (B), (C)

E. All of (A), (B), (C)

> **D — row + diagonal only.** The column constraint is already implied by our choice of variables: there's exactly one x_i per column, so no two queens can ever share a column.

Two ways to express a constraint

Constraint: queens in columns 0 and 2 are not in the same row or diagonal.

As a formula

$$(x_0 \neq x_2) \ \wedge \ (|x_0 - x_2| \neq 2)$$

As a table

x_0	x_2
0	1
0	3
1	0
1	2
2	1
2	3
3	0
3	2

Q: encode (x_0, x_2) as a formula

Q. Which formula encodes "queens in columns 0 and 2 are not in the same row or diagonal"?

A. $(x_0 \neq x_2)$

B. $(x_0 \neq x_2) \ \wedge \ ((x_0 - x_2) \neq 1)$

C. $(x_0 \neq x_2) \ \wedge \ ((x_0 - x_2) \neq 2)$

D. $(x_0 \neq x_2) \ \wedge \ (|x_0 - x_2| \neq 1)$

E. $(x_0 \neq x_2) \ \wedge \ (|x_0 - x_2| \neq 2)$

> **E.** The two columns are 2 apart, so diagonal conflict means $|x_0 - x_2| = 2$. We need both the row and diagonal exclusions, and the absolute value because the queens could be above *or* below each other.

4-queens as an incremental search problem

- **State:** one queen per column in the leftmost k columns, no pair attacking each other.
- **Initial state:** empty board `_ _ _ _`.
- **Goal state:** 4 queens placed, no pair attacking. e.g. `1 3 0 2`.
- **Successor:** place a queen in the leftmost empty column that no current queen attacks.

  e.g. `0 _ _ _` → `0 2 _ _`, `0 3 _ _`.

> Any search algorithm works on this state space. **Backtracking = DFS** + one rule: skip any partial assignment that already violates a constraint.

Backtracking search

Backtracking = DFS through partial assignments + prune any partial that already violates a constraint.

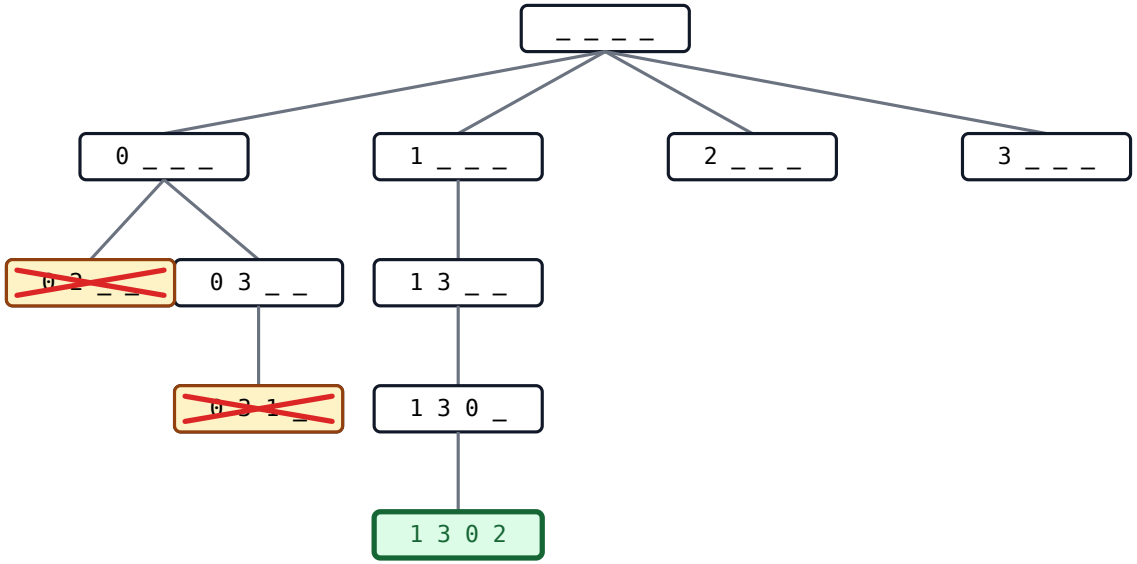
`backtrack(assignment, csp)`

1. If `assignment` is complete → **return** it.
2. Pick an unassigned variable `var`.
3. For each `value` in `domain(var)` that satisfies every constraint:
4. Add `{var = value}` and go deeper; if a solution is found, return it.
5. Otherwise undo and try the next value.
6. If no value worked → **return** `failure`.

Trace: backtracking on 4-queens

Each node = partial assignment. Yellow = dead end. Green = solution.

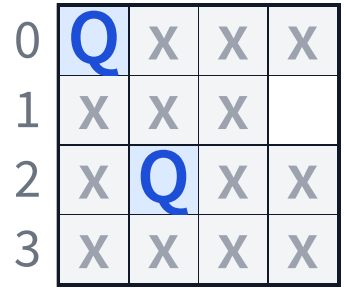
Step:



Found in 9 expansions.

Partial assignment, fatal future

After placing $x_0 = 0$, is $x_1 = 2$ safe?



Every cell of column x_3 is attacked → **no future** for the remaining queens.

Knowing this *before* trying x_2 = **arc consistency**.

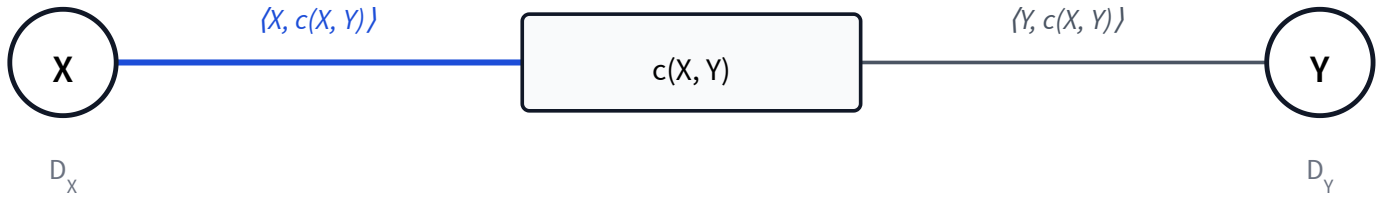
Binary constraints only

Arc consistency is defined on *binary* constraints $c(X, Y)$.

Unary constraints $c(X)$: drop disallowed values directly from D_X.

The CSPs in this course (4-queens, the homework problems) are all binary.

Notation: arcs



- An **arc** $\langle X, c(X, Y) \rangle$ pairs a constraint with one of its variables.
- X is the **primary** variable (the one we'll prune); Y is the **secondary**.
- Each binary constraint defines *two* arcs — one per primary choice.

Arc-consistency definition

Arc consistency. The arc $\langle X, c(X, Y) \rangle$ is *arc-consistent* iff for every $v \in D_X$ there exists a $w \in D_Y$ such that (v, w) satisfies $c(X, Y)$.

Intuition: every value in the primary domain has a partner in the secondary domain.

If some $v \in D_X$ has no partner, remove v from D_X to make the arc consistent.

Q: is this arc consistent?

Q. Constraint $X < Y$, with $D_X = D_Y = \{1, 2\}$. Is $\langle X, X < Y \rangle$ arc-consistent?

A. Yes

B. No

B — No. For $v = 2 \in D_X$, there is no $w \in D_Y$ with $2 < w$. We can **remove 2 from D_X** to make the arc consistent.

The AC-3 algorithm

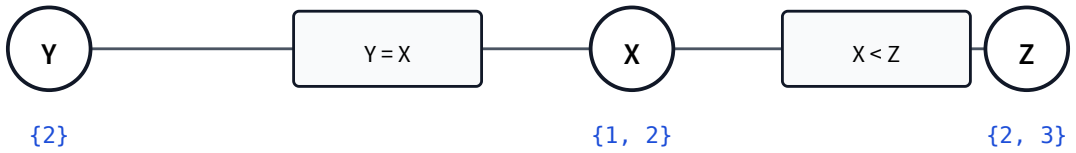
Process arcs one at a time. After each shrink, re-queue neighbours whose consistency may have broken.

`AC-3(arcs)`

1. `S` ← set of all arcs. *initialise the queue*
2. While `S` is not empty:
    1. Pop an arc `<X, c(X, Y)>` from `S`.
    2. For each `v` ∈ D_X with no `w` ∈ D_Y satisfying `c(X, Y)` → remove `v` from D_X.
    3. If D_X is now empty → **return** `false`. *infeasible — no value works*
    4. If D_X shrank → for every neighbour `Z ≠ Y`, add `<Z, c'(Z, X)>` back into `S`. *re-queue — its consistency may have broken*
3. **Return** `true`. *every arc is consistent*

Why re-queue arcs?

Shrinking one domain can break a previously-consistent neighbouring arc.

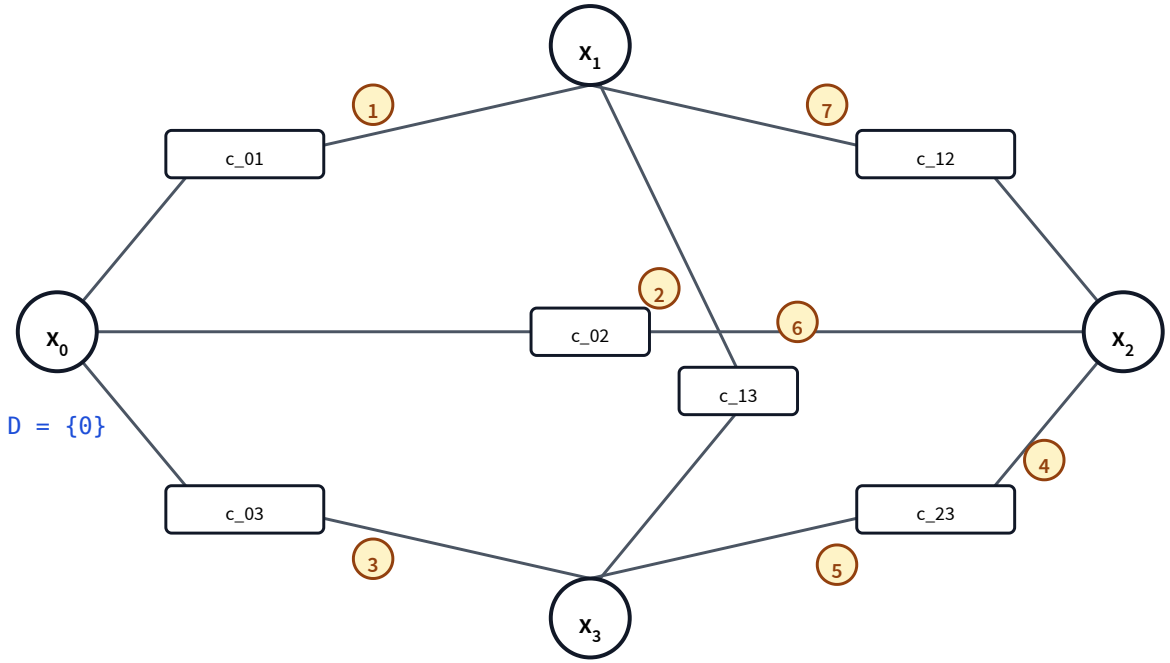


Initially: arc $\langle X, X<Z \rangle$ is consistent (1<2, 1<3, 2<3 all work).

Then: arc $\langle X, Y=X \rangle$ forces $D_X \to \{2\}$.

Now: $\langle Z, X<Z \rangle$ is broken — for $w = 2 \in D_Z$, no $v < 2$ remains in D_X. **Must re-queue.**

Trace: AC-3 on 4-queens with $x_0 = 0$



Three outcomes of AC-3

- **A domain becomes empty** → no solution.
- **Every domain has exactly one value** → solution found, no search needed.
- **Some domains have multiple values** → need backtracking search on what remains.

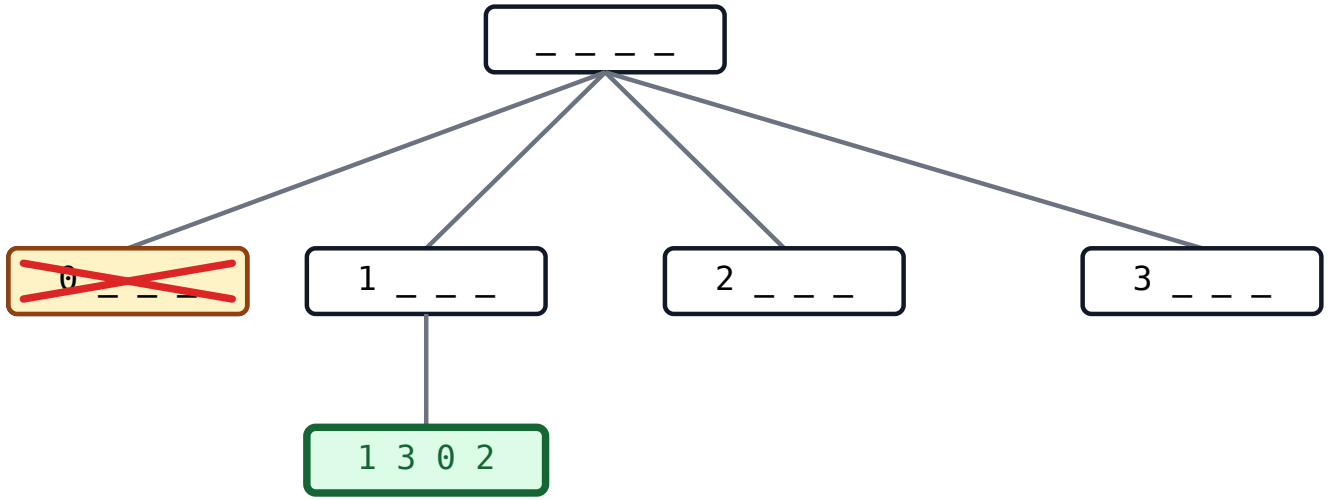
AC-3 — properties

Terminates?	**Yes** *each arc enqueued at most d times*
Order of arcs?	Doesn't matter *same final domains*
Time complexity	$O(c\,d^3)$ *c constraints × d re-queues per arc × $O(d^2)$ per check*

Combining backtracking with AC-3

1. Run backtracking search.
2. After each value assignment, run AC-3.
3. If a domain becomes empty → **fail this branch**, backtrack.
4. If every domain has exactly one value → **solution found**.
5. Otherwise → continue backtracking on the remaining unassigned variables.

Trace: backtracking + AC-3 on 4-queens



Why combine BT with AC-3?

- **AC-3 prunes branches BT would otherwise explore** — it kills bad partial assignments before they grow.
- **BT decides what AC-3 can't** — when AC-3 leaves multi-value domains, BT picks a value and triggers another AC-3 pass.
- On 4-queens with $x_0 = 1$, AC-3 alone reduced everything to a single solution — **zero further search**.

Three techniques for CSPs

Technique	How	Finds a solution?	Best at
Backtracking	DFS through assignments, prune on constraint violation	**Yes**	generic, easy to implement
AC-3 alone	Repeatedly enforce arc consistency	*only if all domains become singletons*	shrinking domains pre-emptively
BT + AC-3	BT picks a value, AC-3 propagates	**Yes** *far fewer expansions*	practical CSP solving

Learning goals (recap)

- ✓ Formulate a real-world problem as a CSP
- ✓ Trace backtracking search
- ✓ Verify arc consistency
- ✓ Trace AC-3
- ✓ Combine backtracking with arc consistency