Artificial Intelligence

Constraint satisfaction problems

The notes for the lesson “Constraint satisfaction problems”: chapters 3.5 to 3.7, with their figures, worked examples, exam traps, quizzes and exam question bank, in one file.

3 chapters | Chapters 3.5 to 3.7

About these notes

These are the notes for the University free n easy video lesson **Constraint satisfaction problems** on Artificial Intelligence: chapters 3.5 to 3.7, with every figure, worked example, exam trap and quiz, and the long-answer exam questions on these chapters.

3.5 Constraint satisfaction problems: variables, domains and constraints

A constraint satisfaction problem (CSP) describes a task by a set of variables, the values each variable may take, and rules that limit which combinations are allowed. A solution gives every variable a value without breaking any rule. Exam timetables, Sudoku, map colouring, seat allotment and chip layout all fit this form, and once a problem is written this way, one general solver (Chapter 3.6) works on all of them.

> You need: states, actions and goal tests from an earlier lesson, *Problem-solving agents and problem formulation*.

The three parts of a CSP

- **X = {X_1, …, X_n}**, a set of variables.
- **D = {D_1, …, D_n}**, one domain per variable: the set of values that variable may take.
- **C**, a set of constraints. Each constraint has a **scope** (the variables it involves) and a **relation** (the value combinations it allows), for example $\langle (X_1, X_2), X_1 \neq X_2 \rangle$.

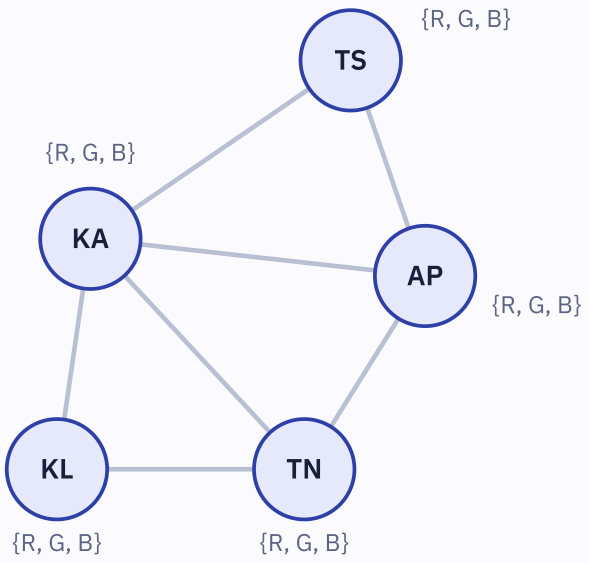
A state of the search is an **assignment** of values to some or all variables. An assignment that breaks no constraint is **consistent** (legal). One that gives every variable a value is **complete**; otherwise it is **partial**. A **solution** is an assignment that is both consistent and complete.

Why bother? In an earlier lesson a state was a black box: the search could only ask "is this the goal?". In a CSP a state has visible parts (a factored representation), so the solver can see which variable breaks which rule and discard every assignment that contains the same mistake at once.

Running example: colouring the South India map. Colour Kerala (KL), Tamil Nadu (TN), Karnataka (KA), Andhra Pradesh (AP) and Telangana (TS) with red, green and blue so that states sharing a border get different colours.

- X = {KL, TN, KA, AP, TS}.
- D_i = {R, G, B} for every state.
- C = {KL ≠ TN, KL ≠ KA, TN ≠ KA, TN ≠ AP, KA ≠ AP, KA ≠ TS, AP ≠ TS}: one rule per shared border, 7 in all.

A binary CSP (every constraint links two variables) is drawn as a **constraint graph**: a node per variable and an edge per constraint (Figure 3.5.1). One solution is KA = R, TN = G, AP = B, KL = B, TS = G.



Variables
X = {KL, TN, KA, AP, TS}

Domains
each D_i = {R, G, B}

Constraints (one per edge)
KL ≠ TN, KL ≠ KA, TN ≠ KA,
TN ≠ AP, KA ≠ AP,
KA ≠ TS, AP ≠ TS

One solution
KA=R, TN=G, AP=B,
KL=B, TS=G

Figure 3.5.1 The South India map as a CSP. KL Kerala, TN Tamil Nadu, KA Karnataka, AP Andhra Pradesh, TS Telangana; nodes are placed roughly where the states lie. Each edge is one ≠ constraint. Notice the triangles KL–KA–TN, TN–KA–AP and KA–AP–TS: each forces three different colours.

Kinds of variables and domains

The **constraint domain** of a problem is the kind of values its variables can take:

- **Discrete, finite domains**: colours, rows of a chessboard, digits 0–9. With n variables and d values each there are d^n complete assignments: $3^5 = 243$ for our map.
- **Discrete, infinite domains**: integers with no upper limit, such as the start day of a job. The allowed pairs cannot be listed in a table, so constraints are written as formulas, for example $T_1 + d_1 \leq T_2$ (job 1, lasting d_1 days, ends before job 2 starts). Linear constraints like this can be solved; no algorithm solves general nonlinear constraints on integers.
- **Continuous domains**: real numbers, such as observation times on a telescope. Linear constraints over reals are solved by linear programming in polynomial time.

Kinds of constraints

- **Unary**: one variable, e.g. TN ≠ G, or "the OS exam is not on Monday".
- **Binary**: two variables, e.g. KA ≠ AP. These are the edges of the constraint graph.
- **Global** (higher-order): any number of variables, e.g. **Alldiff**(X_1, …, X_k), all different. Each row of a Sudoku is an Alldiff on 9 cells, and each column of a cryptarithmetic sum (Chapter 3.7) links three or more letters. These are drawn as a constraint hypergraph.
- **Preference** (soft) constraints: "Prof. Rao prefers morning slots". Breaking one costs something but does not make the assignment illegal. A CSP with costs is a constraint optimisation problem (COP).

Any finite-domain global constraint can be rewritten as binary constraints by adding helper variables, which is why many solvers handle only binary constraints.

WORKED EXAMPLE

Question. Formulate the 4-queens problem (place 4 queens on a 4 × 4 board so that no two attack each other) as a CSP. Give the variables, domains and constraints, count the complete assignments, and check whether (2, 4, 1, 3) and (1, 3, 4, 2) are solutions. (8 marks)

1. **Variables.** Q_1, Q_2, Q_3, Q_4, where Q_i is the row of the queen in column i. One queen per column is built into this choice.
2. **Domains.** $D_i = \{1, 2, 3, 4\}$.
3. **Constraints.** For every pair i < j: $Q_i \neq Q_j$ (not the same row) and $|Q_i - Q_j| \neq j - i$ (not the same diagonal). There are 4C2 = 6 pairs, so 6 binary constraints.
4. **Size.** 4^4 = **256** complete assignments. (Had we used one variable per square with values {queen, empty}, there would be 2^{16} = 65 536. The choice of variables matters.)
5. **Check (2, 4, 1, 3).** Rows 2, 4, 1, 3 are all different. Diagonals, |row difference| against column difference: (1,2): 2 vs 1; (1,3): 1 vs 2; (1,4): 1 vs 3; (2,3): 3 vs 1; (2,4): 1 vs 2; (3,4): 2 vs 1. No pair is equal, so it is consistent and complete: a **solution**.
6. **Check (1, 3, 4, 2).** Columns 2 and 3: |3 − 4| = 1 = 3 − 2, the same diagonal. **Not a solution.**

Answer. $X = \{Q_1, \ldots, Q_4\}$, $D_i = \{1, 2, 3, 4\}$, 6 pairwise row-and-diagonal constraints, 256 complete assignments. Only two of them are solutions: (2, 4, 1, 3) and its mirror image (3, 1, 4, 2).

EXAM TRAPS

- A consistent partial assignment is not a solution; a solution must also be complete.
- The number of complete assignments is d^n (domain size to the power of the number of variables), not n^d or n × d.
- Edges of a constraint graph are constraints, not physical roads; two states that touch only at a single point need no edge.
- A soft constraint never makes an assignment illegal; it only adds a cost.
- Favourite question: "Define a CSP. Formulate map colouring (or 8-queens, or a timetable) as a CSP and draw its constraint graph."

CHECK YOURSELF

- REMEMBER **What are the three components of a CSP?** 2 marks

  A set of variables X, a domain D_i for each variable, and a set of constraints C, each with a scope and a relation.

- REMEMBER **Define a consistent assignment and a complete assignment.** 2 marks

  Consistent: breaks no constraint. Complete: every variable has a value. A solution is both.

- UNDERSTAND **Give one example each of a unary, a binary and a global constraint from a college exam timetable.** 3 marks

  Unary: AI exam not in slot 1. Binary: AI and DBMS (shared students) not in the same slot. Global: Alldiff over all exams of one batch, so no batch has two exams in one slot.

- UNDERSTAND **Why can a CSP solver prune more than the search of an earlier lesson?** 3 marks

  Its states are factored into variables. When a partial assignment breaks a constraint, the solver knows which variables caused it, so it can discard every extension of that partial assignment at once instead of testing each complete state.

- APPLY **How many complete assignments does the 8-queens CSP (one variable per column, domain = rows) have?** 2 marks

  8^8 = **16 777 216**.

- APPLY **Four exams AI, OS, CN, DB; three slots 1–3. Shared students: AI–OS, OS–CN, AI–DB, OS–DB. Write the constraints and find a solution.** 4 marks

  Variables AI, OS, CN, DB, domains {1, 2, 3}; constraints AI ≠ OS, OS ≠ CN, AI ≠ DB, OS ≠ DB. A solution: OS = 1, AI = 2, DB = 3, CN = 2 (CN shares no students with AI).

- ANALYZE **Can the map of Figure 3.5.1 be coloured with two colours? Justify.** 3 marks

  No. KL, KA and TN all border one another (a triangle in the graph), so they need three different colours. Any graph containing a triangle needs at least three colours.

BEYOND THE SYLLABUS

Constraint programming is an industry tool. Google OR-Tools (its CP-SAT solver), MiniZinc and IBM CP Optimizer are used for crew rostering, factory scheduling, university timetabling and sports fixtures. A CSP whose variables are all true/false is the Boolean satisfiability problem (SAT, see a later lesson, propositional logic), and modern SAT solvers handle millions of clauses. Search next for "MiniZinc tutorial" and "CP-SAT". Not for the exam.

3.6 CSP as search: backtracking and constraint propagation

A CSP can be solved as a search problem: start with nothing assigned, give one variable a value at a time, and undo the latest choice as soon as a constraint breaks. This is **backtracking search**, depth-first search over partial assignments. Two things make it fast: choosing the next variable and value with care, and **constraint propagation**, which uses the constraints to shrink the other variables' domains before searching further.

You need: the CSP of Chapter 3.5 and depth-first search (an earlier lesson).

From CSP to search tree

As a search problem: the initial state is the empty assignment {}; an action gives a value to an unassigned variable without breaking a constraint; the goal test is "assignment complete". Every solution lies at depth n, the number of variables.

A careless search tree would allow any of n variables with any of d values at the top (nd branches), (n − 1)d at the next level, and so on: $n! \cdot d^n$ leaves, though only d^n complete assignments exist. But assignments are **commutative**: setting KA = R then TN = G gives the same state as TN = G then KA = R. So backtracking assigns **one variable per level**, and the tree has d^n leaves.

Backtracking search

```
function BACKTRACK(assignment, csp):
    if assignment is complete: return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)      // e.g. MRV
    for each value in ORDER-DOMAIN-VALUES(var, assignment)  // e.g. LCV
        if value is consistent with assignment:
            add {var = value} to assignment
```

```
            inferences ← INFERENCE(csp, var, value)        // e.g. forward checking
            if inferences ≠ failure:
                result ← BACKTRACK(assignment, csp)
                if result ≠ failure: return result
            remove {var = value} and its inferences       // backtrack
    return failure
```

Called first as BACKTRACK({}, csp). Plain backtracking tests each value only against variables already assigned. Figure 3.6.1 shows it on 4-queens.

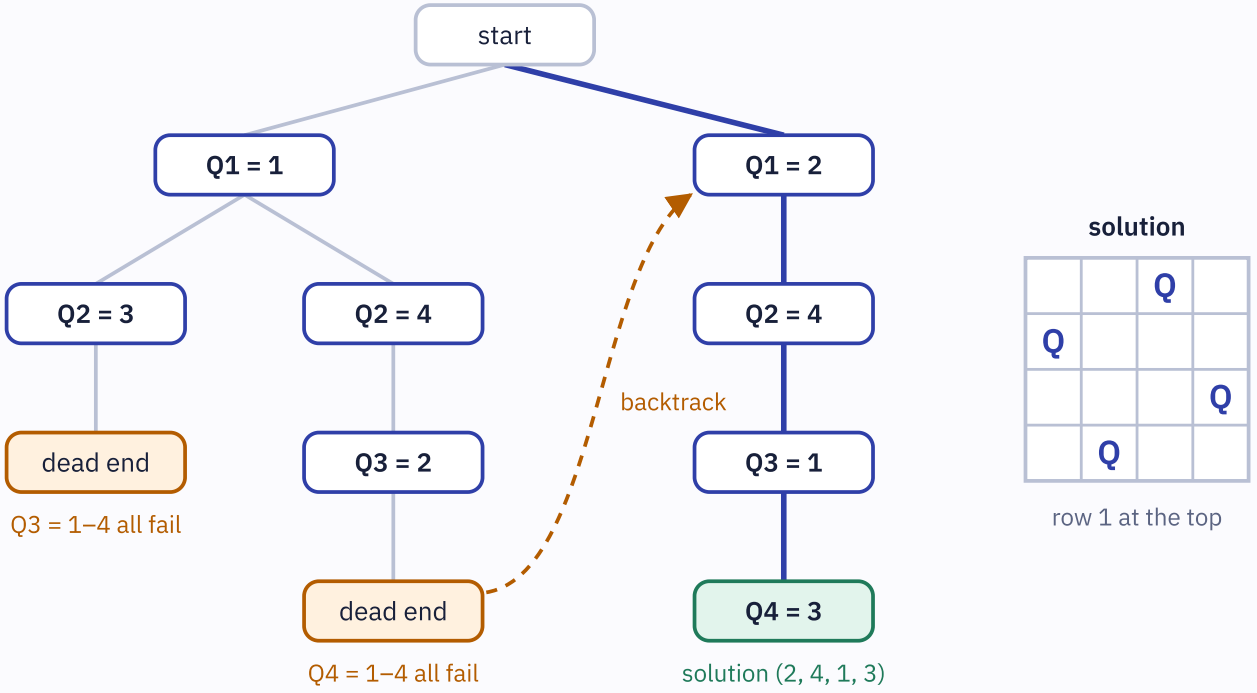


Figure 3.6.1 Plain backtracking on 4-queens, read depth first from left to right. Q_i is the row of the queen in column i. Under Q1 = 1, both branches die, so the search backtracks to the root and tries Q1 = 2, which leads straight to the solution (2, 4, 1, 3). In all, 26 row values are tested.

Choosing the variable and the value

- **Minimum remaining values (MRV)**, also called "most constrained variable" or "fail-first": choose the variable with the fewest legal values left. If some variable has 0 values, the failure is found at once.
- **Degree heuristic**: to break ties (and at the start, when all domains are equal), choose the variable in the most constraints with unassigned variables. In Figure 3.5.1, KA has degree 4, the most, so KA goes first.
- **Least constraining value (LCV)**: for the chosen variable, try first the value that removes the fewest values from its neighbours' domains.

Variables are chosen fail-first, because every variable must be assigned anyway and early failure prunes more; values are chosen succeed-first, because only one solution is needed.

Constraint propagation

- **Node consistency**: every value in D_i satisfies the unary constraints on X_i. Just delete the values that do not.
- **Forward checking (FC)**: after assigning X = x, delete from the domain of every unassigned neighbour Y each value that conflicts with x. If some domain becomes empty, backtrack at once.
- **Arc consistency**: the arc $X_i \to X_j$ is consistent if every value in D_i has at least one supporting value in D_j that satisfies their constraint. **AC-3** (Mackworth, 1977) puts every arc in a queue. It takes an arc (X_i, X_j) and REVISEs it, deleting the values of D_i that have no support. If D_i shrank, it adds back every arc (X_k, X_i) for the other neighbours X_k. It stops when the queue is empty, or reports failure when a domain becomes empty. Its time is $O(c \cdot d^3)$ for c binary constraints and domain size d.
- **MAC** (maintaining arc consistency) runs AC-3 after every assignment. It is stronger than FC, which only checks arcs into the newly assigned variable.

Stronger forms exist (path consistency for triples, k-consistency), and **conflict-directed backjumping** jumps back to the variable that caused a failure instead of the most recent one. Local search can solve CSPs too: the **min-conflicts** heuristic repeatedly moves one variable to the value with the fewest conflicts (compare hill climbing in an earlier lesson).

WORKED EXAMPLE

Question 1. Solve 4-queens by backtracking with forward checking, trying values in increasing order. Show the domains after each assignment. (8 marks)

STEP	ASSIGN	D(Q2)	D(Q3)	D(Q4)	RESULT
0	—	1 2 3 4	1 2 3 4	1 2 3 4	start
1	Q1 = 1	3 4	2 4	2 3	go on
2	Q2 = 3	—	ø	2	D(Q3) empty: undo Q2 = 3
3	Q2 = 4	—	2	3	go on
4	Q3 = 2	—	—	ø	D(Q4) empty; no values left for Q3 or Q2: undo Q1 = 1
5	Q1 = 2	4	1 3	1 3 4	go on
6	Q2 = 4	—	1	1 3	go on
7	Q3 = 1	—	—	3	go on
8	Q4 = 3	—	—	—	solution

For example, at step 1 the queen at row 1, column 1 removes row 1 everywhere and the diagonal squares (row 2 of column 2, row 3 of column 3, row 4 of column 4). **Answer:** (2, 4, 1, 3), reached with only 8 assignments, against 26 values tested by plain backtracking in Figure 3.6.1. FC spots each dead end one level earlier.

Question 2. Variables A, B, C with domains {1, 2, 3} and constraints A < B and B < C. Run AC-3. (5 marks)

1. Queue: (A,B), (B,A), (B,C), (C,B).
2. REVISE(A,B): A = 3 has no larger B, so D(A) = {1, 2}. A has no other neighbour, so nothing is added.
3. REVISE(B,A): B = 1 has no smaller A, so D(B) = {2, 3}. Add (C,B), already queued.
4. REVISE(B,C): B = 3 has no larger C, so D(B) = {2}. Add (A,B).
5. REVISE(C,B): C must exceed 2, so D(C) = {3}.
6. REVISE(A,B): A must be below 2, so D(A) = {1}. Queue empty.

Answer: A = 1, B = 2, C = 3, found by propagation alone with no search.

EXAM TRAPS

- MRV chooses the **variable**; LCV chooses the **value**. Swapping them is the commonest slip.
- When $D(X_i)$ shrinks in AC-3, add the arcs pointing **into** X_i, (X_k, X_i), not the arcs out of it.
- Arc consistency does not prove a solution exists: three mutually adjacent variables with domains {R, G} are arc-consistent, but cannot all differ.
- FC checks only the neighbours of the variable just assigned; it misses conflicts between two unassigned variables, which AC-3 finds.

CHECK YOURSELF

– REMEMBER **What does MRV stand for, and what are its other names?** 1 mark

Minimum remaining values; also "most constrained variable" and "fail-first".

– REMEMBER **State the time complexity of AC-3.** 1 mark

$O(c \cdot d^3)$, c binary constraints, domain size d.

– UNDERSTAND **Why does backtracking assign only one variable per level of the tree?** 3 marks

Assignments are commutative: the order in which values are given does not change the final assignment. Fixing one variable per level cuts the leaves from $n! \cdot d^n$ to d^n without losing any solution.

– APPLY **Colour the map of Figure 3.5.1 by backtracking with degree, MRV and forward checking (colours tried in order R, G, B).** 5 marks

KA (degree 4) = R; FC leaves {G, B} for TN, KL, AP, TS. Tie on MRV; degree among unassigned: TN 2 (KL, AP), AP 2, KL 1, TS 1, so TN = G; now KL {B}, AP {B}. MRV: AP = B, so TS {G}. KL = B, TS = G. **No backtracking needed.**

– APPLY **D(X) = D(Y) = {1, 2, 3, 4} and X = 2Y. Make both arcs consistent.** 2 marks

X must be twice some Y: D(X) = {2, 4}. Y must be half of some X: D(Y) = {1, 2}.

– ANALYZE **Compare forward checking and MAC on the cost per node and the size of the tree.** 4 marks

FC does little work per node (only arcs into the new variable) but misses some conflicts, so the tree is larger. MAC propagates through the whole graph after each assignment, costs more per node, and detects failures earlier, giving a smaller tree. On hard problems MAC usually wins overall.

BEYOND THE SYLLABUS

Modern SAT solvers use conflict-driven clause learning (CDCL): when they hit a dead end they record the reason as a new constraint, so the same mistake is never repeated; this grew out of backjumping. Constraint solvers combine propagation with search and with linear programming. Search next for "CDCL SAT solver" and "min-conflicts n-queens". Not for the exam.

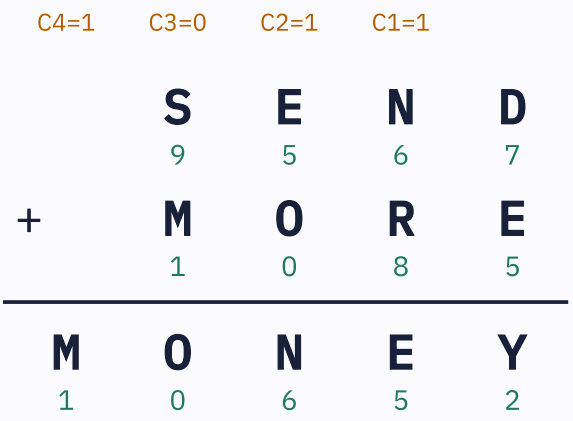
3.7 Cryptarithmetic and map colouring problems

Two classic puzzles that are solved by hand as CSPs. In a **cryptarithmetic** puzzle each letter stands for a different digit and the sum must be correct. In **map colouring** (or room colouring) neighbours must get different colours. Both are common exam questions wherever CSPs are taught.

Cryptarithmetic as a CSP

- **Variables:** the letters, plus one carry variable C_1, C_2, ... per column.
- **Domains:** {0, ..., 9} for letters; {0, 1} for carries when two numbers are added.
- **Constraints:** Alldiff over the letters; leading letters ≠ 0; one column equation per column: digits + carry in = result digit + 10 × carry out.

By hand, start from the leftmost column, where domains are smallest (MRV again), and put each value found into every equation.



Column equations (carries 0 or 1)

units: D + E = Y + 10·C1

tens: N + R + C1 = E + 10·C2

hundreds: E + O + C2 = N + 10·C3

thousands: S + M + C3 = O + 10·C4

ten-thousands: C4 = M

Alldiff(S, E, N, D, M, O, R, Y)

S ≠ 0, M ≠ 0

9567 + 1085 = 10652

Figure 3.7.1 SEND + MORE = MONEY as a CSP. Orange: carries, above the column they carry into. Green: each letter's digit in the unique solution.

WORKED EXAMPLE

Question. Solve SEND + MORE = MONEY. (8 marks)

1. **Ten-thousands:** M = C4, and M ≠ 0, so **M = 1**.
2. **Thousands:** S + 1 + C3 = O + 10. With S ≤ 9, O ≤ 1; 1 is taken, so **O = 0**, and S = 9 − C3.
3. **Hundreds:** E + 0 + C2 = N + 10·C3. If C3 = 1, then E + C2 ≥ 10 forces E = 9, C2 = 1, N = 0, but O = 0. So **C3 = 0** and **S = 9**. Then E + C2 = N; since N ≠ E, **C2 = 1** and N = E + 1.
4. **Tens:** N + R + C1 = E + 10. Put N = E + 1: R + C1 = 9. R = 9 is taken by S, so **C1 = 1** and **R = 8**.
5. **Units:** D + E = Y + 10. Digits left: {2, 3, 4, 5, 6, 7}, and Y ≥ 2 means D + E ≥ 12. Since D = Y + 10 − E, the values E = 2, 3, 4 need D ≥ 10, 9 or 8, and 8 and 9 are taken. E = 5, N = 6: D = Y + 5 gives **Y = 2, D = 7** (Y = 3 or 4 would need D = 8 or 9). E = 6, N = 7 fails (D would clash with 6, 7, 8 or 9).

Answer. S = 9, E = 5, N = 6, D = 7, M = 1, O = 0, R = 8, Y = 2. Check: 9567 + 1085 = 10652. The solution is unique.

Map and room colouring

The South India map of Chapter 3.5 is the standard map-colouring CSP, solved in Chapter 3.6. **Room colouring** is the same problem on a floor plan: rooms sharing a wall must differ. Say hall H shares walls with kitchen K, bathroom Ba and bedrooms B1, B2, and K–Ba, Ba–B2, B2–B1 also share walls. Degree picks H first (degree 4): H = red. The chain K–Ba–B2–B1 then needs two colours: K = green, Ba = blue, B2 = green, B1 = blue. Three colours are needed, since H, K, Ba form a triangle. The least number is the **chromatic number**; by the four colour theorem (Appel and Haken, 1976) four always suffice for a flat map.

EXAM TRAPS

- Forgetting the carries, or giving them the domain 0–9; for two numbers a carry is 0 or 1.
- The extra leading digit of a two-number sum is always 1.
- Show the reasoning column by column; a bare answer earns few marks.

CHECK YOURSELF

– REMEMBER **List the constraints of a cryptarithmetic CSP.** 2 marks

Alldiff over the letters, leading letters non-zero, and one column equation per column with carries.

– UNDERSTAND **Why must M = 1 in SEND + MORE = MONEY?** 2 marks

M is the carry out of a sum of two 4-digit numbers, so M ≤ 1, and as a leading digit M ≠ 0.

– APPLY **Solve TWO + TWO = FOUR given T = 7 and O = 4.** 4 marks

Units: 4 + 4 = 8, so R = 8, carry 0. Hundreds: 7 + 7 + carry = 14 + carry gives O = 4 only if carry = 0 and F = 1. Tens (carry out 0): U = 2W, so W ≤ 4. W = 0 gives U = W; 1 and 4 are taken (F, O); W = 2 gives U = 4 = O. So W = 3, U = 6: **734 + 734 = 1468.**

– APPLY **In the flat above, K and B1 also share a wall. How many colours are needed now?** 2 marks

Still **3**: H = red; K–Ba–B2–B1–K is a 4-cycle, which two colours cover (K green, Ba blue, B2 green, B1 blue).

– ANALYZE **Why is it efficient to start a cryptarithmetic from the leftmost column?** 3 marks

That column has the fewest possibilities (M has only the value 1), which is the MRV idea. Fixing it fixes O and S, which then propagate to the other columns.

BEYOND THE SYLLABUS

Compilers colour an interference graph to assign registers, and mobile networks colour cells to assign frequencies. Deciding whether a graph can be 3-coloured is NP-complete. Search next for "register allocation graph colouring". Not for the exam.

Key terms of this lesson

TERM	MEANING OR FORMULA	CHAPTER
CSP	variables X, domains D, constraints C; solution = consistent and complete assignment	3.5
Complete assignments	d^n for n variables with d values each	3.5
MRV, degree, LCV	fewest values left; most constraints on unassigned variables; value that rules out least	3.6
Forward checking, AC-3	prune neighbours of the assigned variable; make every arc consistent, $O(c \cdot d^3)$	3.6
Column equation	digits + carry in = result digit + 10 × carry out	3.7

Exam question bank

Long-answer exam questions on these chapters, each with an outline of the points an examiner looks for. Answer one in full on paper before you read its outline.

Remember and understand

– Q3 **Define a constraint satisfaction problem. Explain the kinds of domains and constraints with examples, and draw the constraint graph of a map-colouring problem.** 10 marks

Points: X, D, C with scope and relation; consistent, complete, solution; finite, infinite and continuous domains; unary, binary, global (Alldiff), soft constraints; constraint graph such as Figure 3.5.1; d^n assignments.

Apply

– Q6 **Solve the cryptarithmetic BASE + BALL = GAMES. Show the column equations and the reasoning.** 10 marks

G = carry = 1. Adding the units and tens equations gives $2L = 10c_2 + 9c_1$, so $c_1 = 0$ and L = 5, $c_2 = 1$ (L = 0 would make E = S). Units: S = E + 5. Hundreds $2A + 1 = M + 10c_3$, thousands $2B + c_3 = A + 10$: only B = 7, A = 4, M = 9, $c_3 = 0$ survive. Then E = 3, S = 8. **7483 + 7455 = 14938.**

– Q7 **Colour the map of Australia (WA, NT, SA, Q, NSW, V, T; borders WA–NT, WA–SA, NT–SA, NT–Q, SA–Q, SA–NSW, SA–V, Q–NSW, NSW–V; T is an island) with red, green, blue using backtracking with the degree heuristic, MRV and forward checking.** 10 marks

SA has degree 5: SA = R; FC removes R from WA, NT, Q, NSW, V. Tie: NT, Q and NSW each have 2 unassigned neighbours; take NT = G, so WA = {B}, Q = {B}. MRV: Q = B, so NSW = {G}; NSW = G, so V = {B}; V = B; WA = B; T = R (no constraint). No backtracking. Draw the constraint graph and a domain table.

– Q9 **Variables X, Y, Z with domains {1, 2, 3, 4} and constraints X > Y and Y > Z. Run AC-3, give the final domains and say whether search is still needed.** 8 marks

REVISE(X,Y) removes 1; (Y,X) removes 4; (Y,Z) removes 1; (Z,Y) removes 3, 4; re-check (X,Y) removes 2. Final: X {3, 4}, Y {2, 3}, Z {1, 2}. Arc-consistent but not solved: four solutions remain, (3,2,1), (4,2,1), (4,3,1), (4,3,2), so backtracking picks one.

Analyze and evaluate

– Q11 **"Alpha-beta pruning and constraint propagation both cut the search tree." Compare the two. (spans Chapters 3.3 and 3.6)** 10 marks

Points: both work inside depth-first search; alpha-beta uses bounds on values (an optimisation against an opponent), propagation uses constraints on domains (a feasibility problem); alpha-beta never changes the answer, propagation never removes a solution; both depend on ordering (moves; variables and values); both find failures early; alpha-beta works after values are known, FC and AC-3 work before the next choice.

– Q14 **Formulate your college's end-semester exam timetable as a CSP and say which heuristics and propagation you would use. (spans Chapters 3.5–3.6)** 10 marks

Points: variables = exams; domains = slots (day and session), possibly halls; hard constraints: exams sharing students differ, hall capacity, unary "not before day 3"; soft: gaps for a batch; constraint graph; MRV and degree (exams with most shared students first), LCV for slots, FC or MAC; min-conflicts for large instances; COP for soft constraints.

Glossary (terms used in this lesson)

Arc consistency	Every value of X_i has a supporting value in X_j for their constraint.
Assignment	Values given to some (partial) or all (complete) variables of a CSP; consistent if no constraint is broken.
Backtracking search	Depth-first search that assigns one variable per level and undoes the last choice on failure.
Chromatic number	The least number of colours that colours a graph with no two neighbours alike.
Constraint graph	A graph with a node per variable and an edge per binary constraint.
Constraint propagation	Using constraints to remove values from domains before or during search.
CSP	Constraint satisfaction problem: variables, domains and constraints.
Cryptarithmetic	A puzzle in which letters stand for distinct digits in a correct sum.
Degree heuristic	Pick the variable in the most constraints with unassigned variables.
Domain	The set of values a variable may take.
Forward checking	After an assignment, delete conflicting values from the neighbours' domains.
LCV	Least constraining value: try first the value that rules out the fewest neighbour values.
MRV	Minimum remaining values: pick the variable with the fewest legal values left.
Ply	One move by one player; one level of the game tree.

Sources and further reading

Textbooks

- Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*, 4th edition, Pearson: the chapters "Adversarial Search and Games" and "Constraint Satisfaction Problems" (chapters 5 and 6 in the 3rd and 4th editions).
- Elaine Rich, Kevin Knight and Shivashankar B. Nair, *Artificial Intelligence*, 3rd edition, McGraw Hill: the chapters on game playing and on constraint satisfaction, including cryptarithmetic.
- Saroj Kaushik, *Artificial Intelligence*, Cengage Learning: game playing with bounded look-ahead and evaluation functions, alpha-beta pruning, and constraint satisfaction.
- Donald E. Knuth and Ronald W. Moore, "An analysis of alpha-beta pruning", *Artificial Intelligence*, 1975: the best-case leaf count used in Chapter 3.3.
- Alan K. Mackworth, "Consistency in networks of relations", *Artificial Intelligence*, 1977: the AC-3 algorithm.

Open web resources

The network was closed while this book was written, so these pages could not be opened; the links are given from memory. Every tree, table and puzzle answer in the book was instead checked by running small Python programs (minimax, alpha-beta, forward checking and brute-force puzzle search), and the program outputs shown are the real ones.

- AIMA code repository, Python version (`games.py`, `csp.py`): github.com/aimacode/aima-python.

Syllabus: the University free n easy universal syllabus, built from top universities' syllabi.

Built for the University free n easy YouTube channel from the universal syllabus of Artificial Intelligence (Oct 2026). Written with top universities' syllabi and standard textbooks; check a formula against your prescribed book before an exam, and report mistakes on the channel.