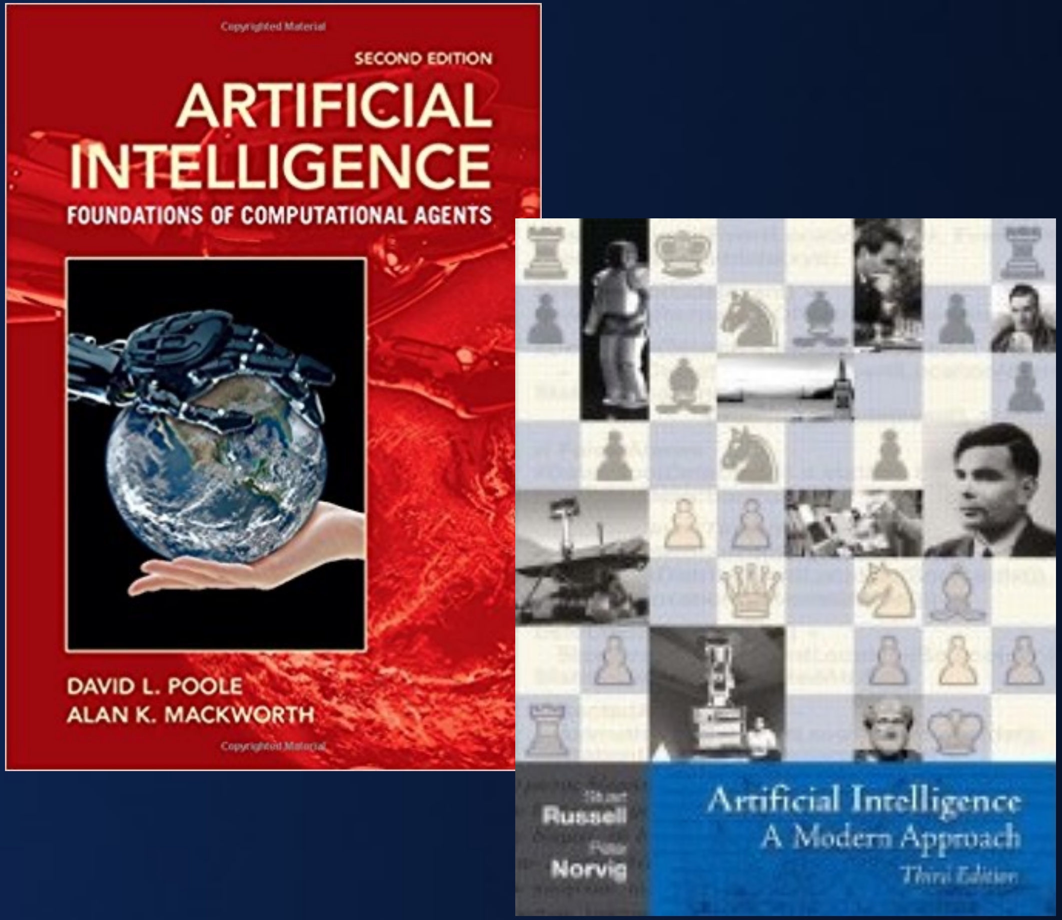


AI Fundamentals: Constraints Satisfaction Problems

Maria Simi

The structure of problems

LESSON 4: THE STRUCTURE OF PROBLEMS (AIMA, CH 6)

Independent sub-problems

When problems have a specific structure, reflected in properties of the constraint graph, there are strategies for improving the process of finding a solution.

A first obvious case is that of **independent subproblems**.

Example:

In the map coloring example, Tasmania is not connected to the mainland; coloring Tasmania and coloring the mainland are independent sub-problems—any solution the mainland combined with any solution for Tasmania yields a solution for the whole map.

Each **connected component** of the constraint graph corresponds to a sub-problem CSP_i.

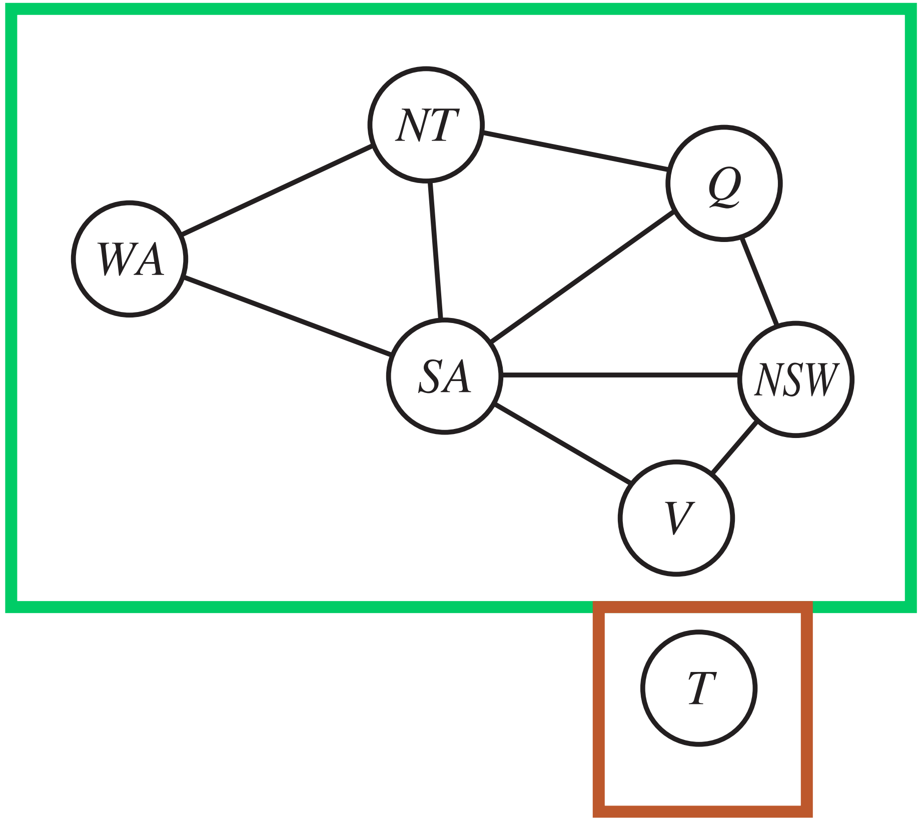
If assignment S_i is a solution of CSP_i, then $\bigcup_i S_i$ is a solution of $\bigcup_i CSP_i$

Independent sub-problems: complexity

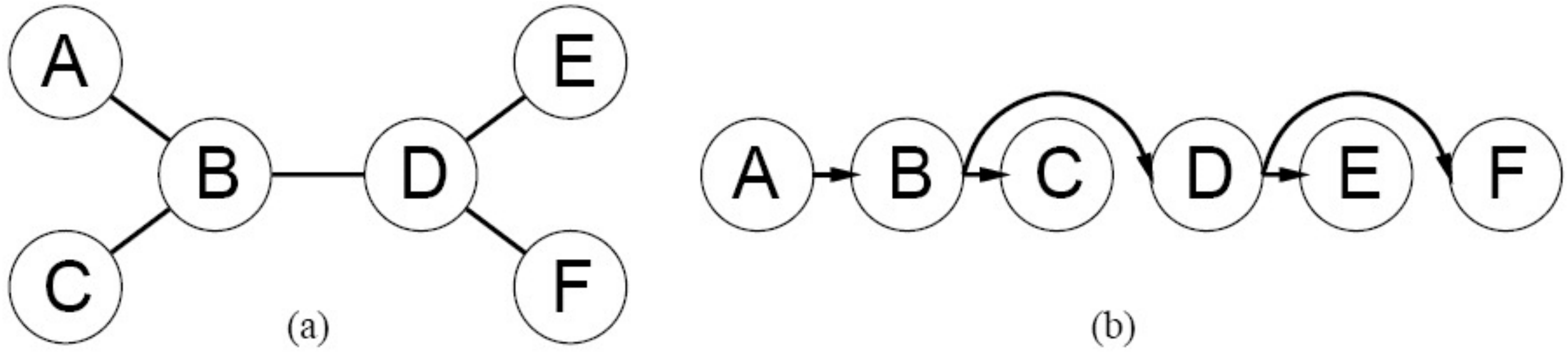
The saving in computational time is dramatic

- n # variables
- c # variables for sub-problems
- n/c independent problems
- d size of the domain
- $O(d^c)$ complexity of solving one
- $O(d^c\, n/c)$ *linear* on the number of variables n rather than $O(d^n)$ *exponential*!

Dividing a Boolean CSP with 80 variables into four sub-problems reduces the worst-case solution time from the lifetime of the universe down to less than a second!!!



The structure of problems: trees



a) In a tree-structured constraint graph, two nodes are connected by only one path; we can choose any variable as the root of a tree. A in fig (b).
b) Chosen a variable as the root, the tree induces a **topological sort** on the variables. Children of a node are listed after their parent.

Directional Arc Consistency (DAC)

A CSP constraint graph is defined to be **directional arc-consistent** under an ordering of variables $X_1, X_2, \dots X_n$ if and only if every X_i is arc-consistent with each X_j for $j > i$.

We can make a tree-like-graph *directional arc-consistent* in one pass over the *n variables*; each step must compare up to d possible domain values for two variables (d^2) for a total time of $O(nd^2)$.

Tree-CSP-solver:

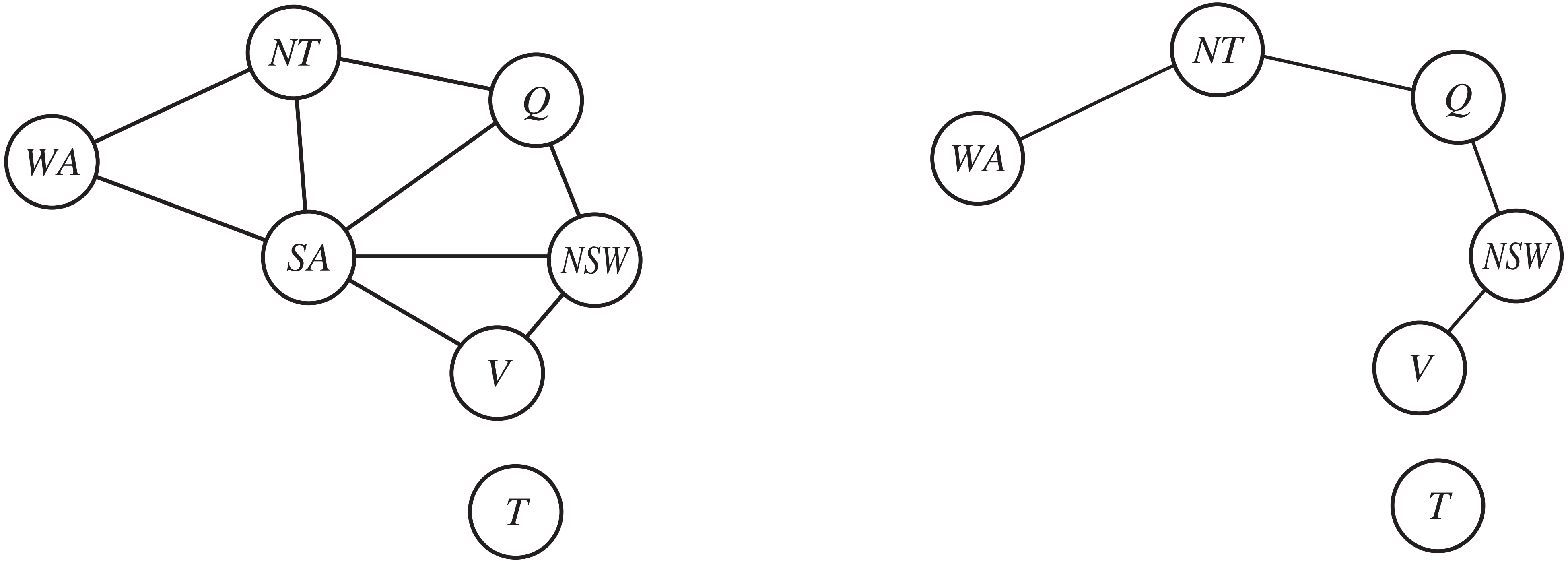
1. Proceeding from X_n to X_2 make the arcs $X_i \rightarrow X_j$ *DAC* consistent by reducing the domain of X_i, if necessary. It can be done in one pass.
2. Proceeding from X_1 to X_n assign values to variables; *no need for backtracking* since each value for a father has at least one legal value for the child.

Tree-CSP-Solver algorithm

```
function TREE-CSP-SOLVER(csp) returns a solution, or failure
  inputs: csp, a CSP with components X, D, C

  n ← number of variables in X
  assignment ← an empty assignment
  root ← any variable in X
  X ← TOPOLOGICALSORT(X, root)
  for j = n down to 2 do
     MAKE-ARC-CONSISTENT(PARENT(X_j), X_j)
     if it cannot be made consistent then return failure
  for i = 1 to n do
     assignment[X_i] ← any consistent value from D_i
     if there is no consistent value then return failure
  return assignment
```

Reducing graphs to trees



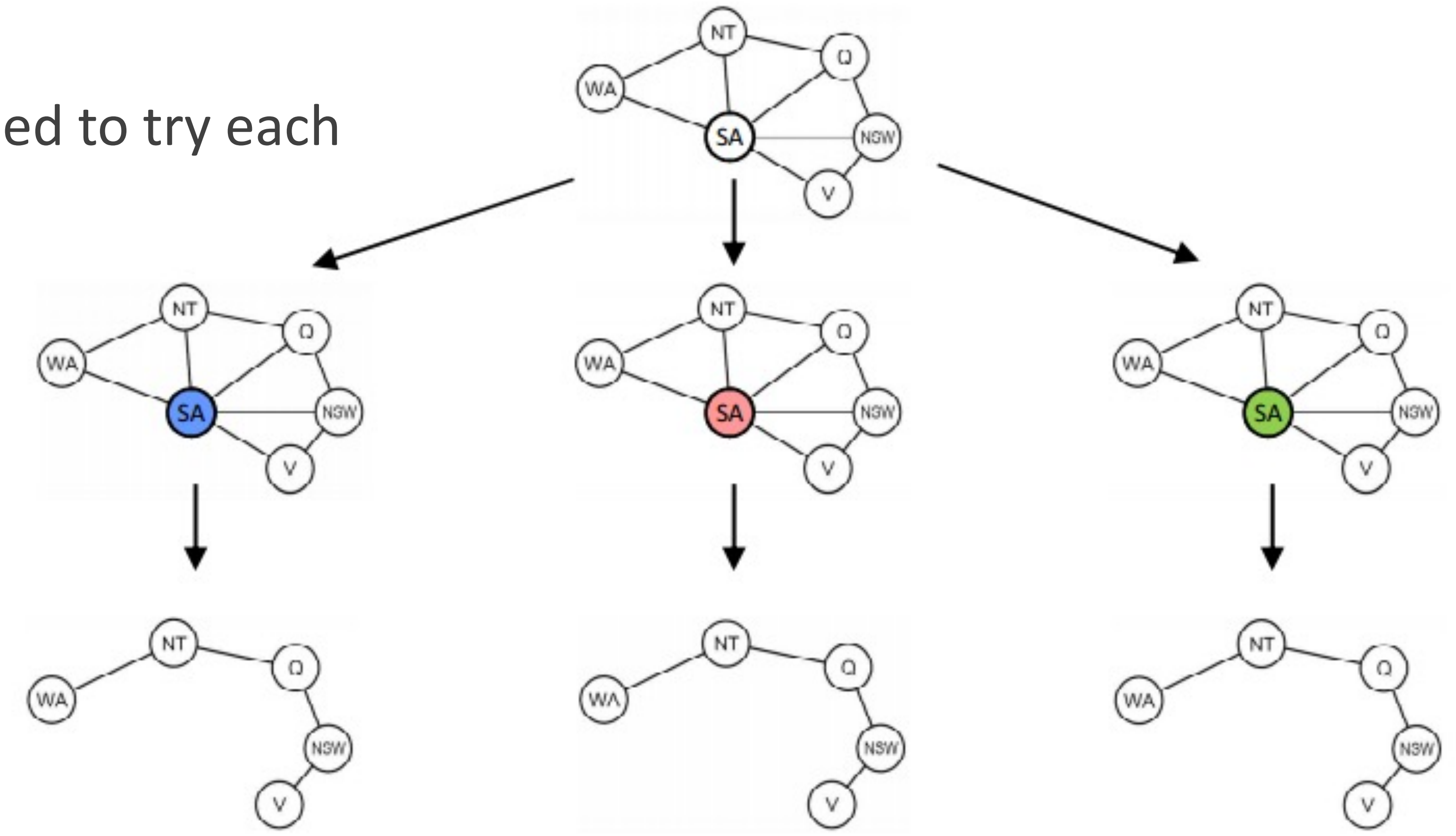
If we could delete South Australia, the graph would become a tree.

This can be done by establishing a value for SA and removing inconsistent values from the other variables. Then solve with Tree-CSP-solver.

Cutset conditioning

In map coloring the color does not really matter

In general we would need to try each possible value.



Cutset conditioning

In general, we must apply a **domain splitting** strategy, trying with different assignments:

1. Choose a subset S of the CSP's variables such that the constraint graph becomes a tree after removal of S. S is called a **cycle cutset**.
2. For each possible consistent assignment to the variables in S:
   a. remove from the domains of the remaining variables any values that are inconsistent with the assignment for S
   b. If the remaining CSP has a solution, return it together with the assignment for S.

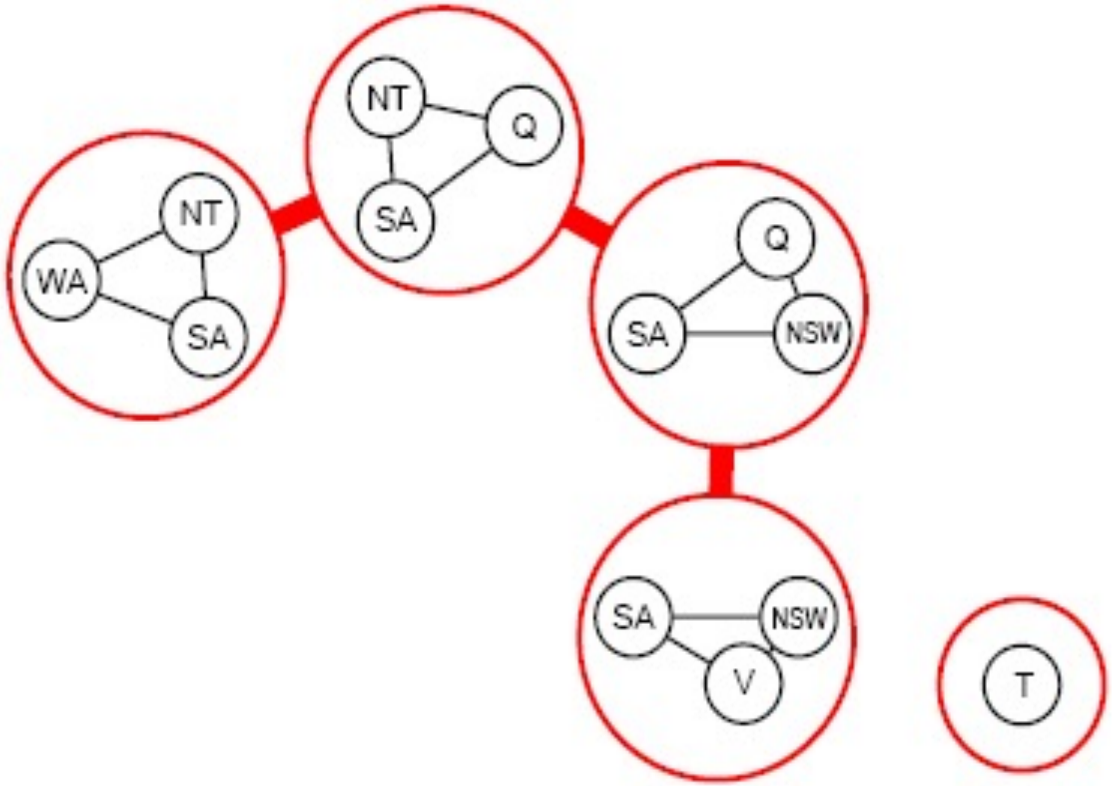
Time complexity: O$(d^c\,(n-c)\,d^2)$
where c **is the size of the cycle cutset** and d the size of the domain

We have to try each of the d^c combinations of values for the variables in S, and for each combination we must solve a tree problem of size $(n-c)$.

Tree decomposition

The approach consists in a **tree decomposition** of the constraint graph into a set of connected sub-problems.

Each sub-problem is solved independently, and the resulting solutions are then combined in a clever way



Properties of a tree decomposition

A tree decomposition must satisfy the following three requirements:

1. Every variable in the original problem appears in at least one of the sub-problems.
2. If two variables are connected by a constraint in the original problem, they must appear together (along with the constraint) in at least one of the sub-problems.
3. If a variable appears in two sub-problems in the tree, it must appear in every subproblem along the path connecting those sub-problems.

Conditions 1-2 ensure that all the variables and constraints are represented in the decomposition.

Condition 3 reflects the constraint that any given variable must have the same value in every sub-problem in which it appears; the links joining sub-problems in the tree will enforce this constraint.

Solving a decomposed problem

- We solve each sub-problem independently. If any problem has no solution, the original problem has no solution.
- Putting solutions together. We solve a meta-problem defined as follows:
  - Each sub-problem is "mega-variable" whose domain is the set of all solutions for the sub-problem

    Ex. *Dom*(X_1) ={⟨WA=r, SA=b, NT=g⟩ ...} the 6 solutions to first subproblem
  - The constraints ensure that the subproblem solutions assign the same values to the the variables they share.

Tree-width of a decomposition: the size of the largest sub-problem – 1.

Ideally we should find, among many possible ones, a tree decomposition with **minimal tree width**. This is NP-hard but heuristics exist.

Simmetry

Simmetry is an important factor for reducing the complexity of CSP problems.

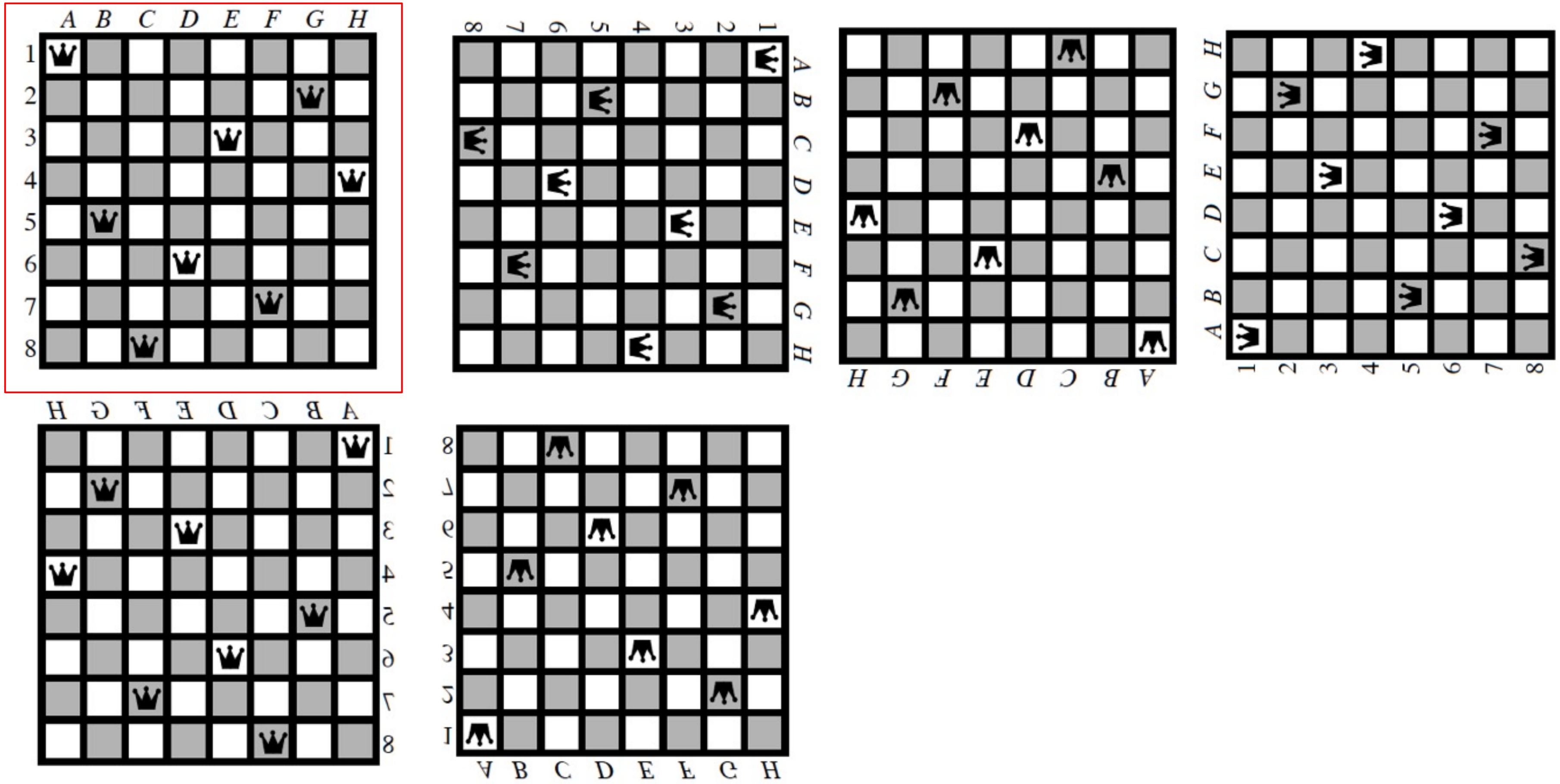
Value simmetry: the value does not really matter.

- *WA*, *NT*, and *SA* must all have different colors, but there are 6 equivalent ways to satisfy the constraints. If *S* is a solution to the map coloring with *n* variables, there are *n*! solutions formed by permuting the color names.

Symmetry-breaking constraints:

- we might impose an arbitrary ordering constraint, $NT < SA < WA$, that requires the three values to be in alphabetical order; we get only one solution out of the 6.
- In practice, *breaking value symmetry* has proved to be important and effective on a wide range of problems.

One solution, many solutions



Rotate 90, 180, 270 left

Flip horizontal | Flip vertical | Flip diagonal 1 | Flip diagonal 2

Three approaches

Three main approaches to symmetry breaking:

1. Reformulate the problem so that it has a reduced amount of symmetry, or even none at all.
2. Add symmetry breaking constraints before search starts making some symmetric solutions unacceptable while leaving at least one solution in each symmetric equivalence class.
3. Break symmetry dynamically during search

A very active area of research …

Conclusions

- ✓ We have seen how to we can exploit the structure of the problem to simplify the algorithms
- ✓ Ideas to reduce a general constraint graph to a graph with a nicer structure
- ✓ CSP is a large field of study: many more things could be presented
- ✓ We did not deal with optimization techniques: the boundaries with Operations research

References

[AIMA] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach* (3rd edition). Pearson Education 2010. (Cap. 6)

[AIFCA] David L. Poole, Alan K. Mackworth. *Artificial Intelligence: foundations of computational agents* (2nd edition), Cambridge University Press, 2017– Computers. http://artint.info/2e/html/ArtInt2e.html (Cap 4)

[Tsang] Edward Tsang. *Foundations of Constraint Satisfaction*, Computation in Cognitive Science. Elsevier Science. Kindle Edition, 2014.