Artificial Intelligence

9. CSP, Part I: Basics, and Naïve Search

What to Do When Your Problem is to Satisfy All These Constraints

Jörg Hoffmann Jana Koehler

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Online (Summer) Term 2020

Agenda

A (Constraint Satisfaction) Problem

→ Who's going to play against who, when and where?

Constraint Satisfaction Problems

What is a constraint?

A constraint is a condition that every solution must satisfy.

What is a constraint satisfaction problem?

Given:

- A set of variables, each associated with its domain.
- A set of constraints over these variables.

Find:

- An assignment of variables to values (from the respective domains), so that every constraint is satisfied.

A Constraint Satisfaction Problem

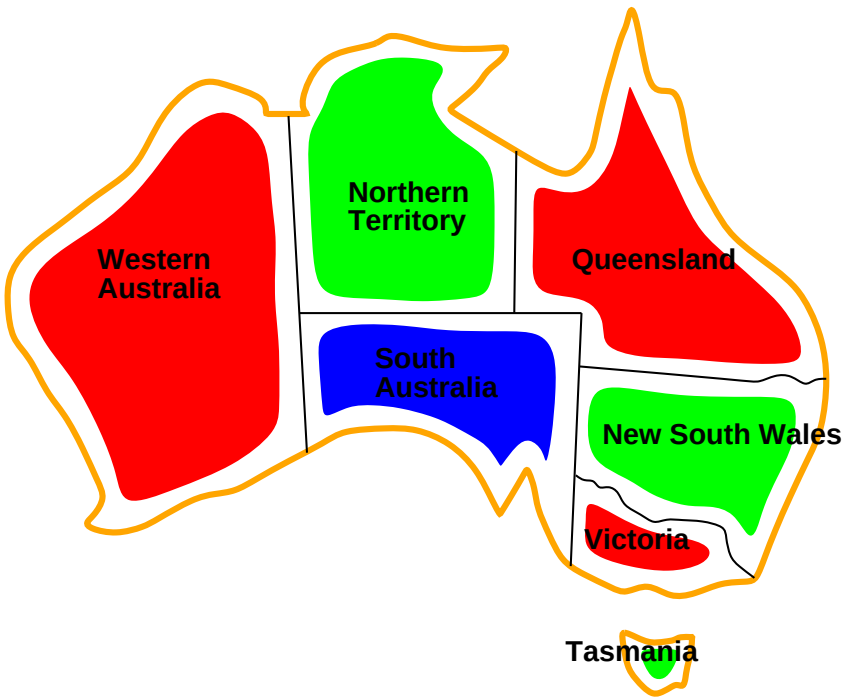
→ Problem: SuDoKu.

2	5	8	7	3	6	9	4	1
6	1	9	8	2	4	3	5	7
4	3	7	9	1	5	2	6	8
3	9	5	2	7	1	4	8	6
7	6	2	4	9	8	1	3	5
8	4	1	6	5	3	7	2	9
1	8	4	3	6	9	5	7	2
5	7	6	1	4	2	8	9	3
9	2	3	5	8	7	6	1	4

- **Variables:** Content of each cell.
- **Domains:** Numbers $1, \ldots, 9$.
- **Constraints:** Each number only once in each row, column, block.

Another Constraint Satisfaction Problem

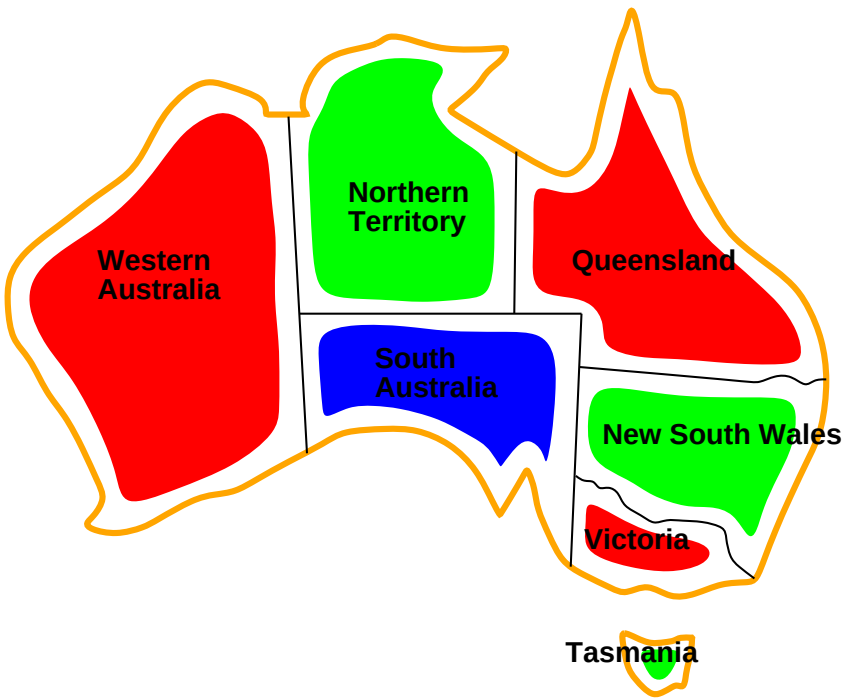
→ (Our Main Illustrative) Problem: Coloring Australia.



- **Variables:** WA, NT, SA, Q, NSW, V, T.
- **Domains:** red, green, blue.
- **Constraints:** Adjacent states must have different colors.

Another Constraint Satisfaction Problem

$\rightarrow$ Problem: Graph Coloring. **NP**-hard for $k = 3$.



- **Variables:** Vertices in a graph.
- **Domains:** k different colors.
- **Constraints:** Adjacent vertices must have different colors.

Bundesliga Constraints

Variables: $v_{A\text{vs.}B}$ where A and B are teams, with domain $\{1, \ldots, 34\}$: For each match, the (ID of the) "Spieltag" where it is scheduled.

(Some) Constraints:

- For all A, B: $v_{A\text{vs.}B} \leq 17 < v_{B\text{vs.}A}$ or $v_{B\text{vs.}A} \leq 17 < v_{A\text{vs.}B}$ (each pairing exactly once in each half-season).
- For all A, B, C, D where $\{A, B\} \cap \{C, D\} \neq \emptyset$: $v_{A\text{vs.}B} \neq v_{C\text{vs.}D}$ (each team only one match per day).
- For all A, B, D: $v_{A\text{vs.}B} + 1 \neq v_{A\text{vs.}D}$ (each team alternates between home matches and away matches).
- ...

How to Solve the Bundesliga Constraints?

My personal pre-study attempts:

1. 306 nested for-loops (for each of the 306 matches), each ranging from 1 to 306. Within the innermost loop, test whether the current values are (a) a permutation and, if so, (b) a legal Bundesliga schedule.

   $\rightarrow$ Estimated runtime (on a Commodore 128): End of this universe, and the next couple million ones after it ...

2. Directly enumerate all permutations of the numbers $1, \ldots, 306$, test for each whether it's a legal Bundesliga schedule.

   $\rightarrow$ Estimated runtime: Maybe only the time span of a few thousand universes.

3. View this as variables/constraints and use backtracking (**This Chapter**).

   $\rightarrow$ Executed runtime: About 1 minute.

How do they actually do it? Modern computers and CSP methods: fractions of a second. 19th (20th/21st?) century: Combinatorics and manual work.

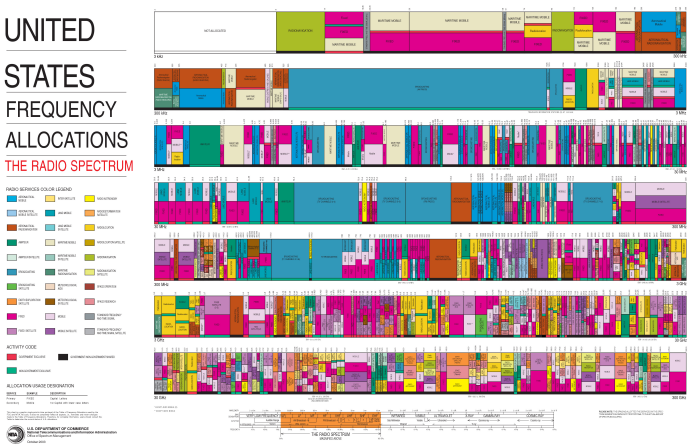
Some Applications

Traveling Tournament Problem

Scheduling

Timetabling

Radio Frequency Assignment



Our Agenda for This Topic

→ Our treatment of the topic "Constraint Satisfaction Problems" consists of Chapters 9 and 10.

- **This Chapter:** Basic definitions and concepts; naïve backtracking search.

  → Sets up the framework. Backtracking underlies many successful algorithms for solving constraint satisfaction problems (and, naturally, we start with the simplest version thereof).

- **Chapter 10:** Inference and decomposition methods.

  → Inference reduces the search space of backtracking. Decomposition methods break the probem into smaller pieces. Both are crucial for efficiency in practice.

Our Agenda for This Chapter

- **Constraint Networks** and **Assignments, Consistency, Solutions:** How are constraint satisfaction problems defined? What is a solution?

  → Get ourselves on firm ground.

- **Naïve Backtracking:** How does backtracking work? What are its main weaknesses?

  → Serves to understand the basic workings of this wide-spread algorithm, and to motivate its enhancements.

- **Variable- and Value Ordering:** How should we give direction to a backtracking search?

  → Simple methods for making backtracking aware of the structure of the problem, and thereby reduce search.

Constraint Networks: Informal

Constraint Networks: Informal Definition

A constraint network is defined by:

- A finite set of variables.
- A finite domain for each variable.
- A set of constraints (here: binary relations).

$\rightarrow$ We're looking for a solution to the network, i.e., an assignment of variables to values (from the respective domains), so that every constraint is satisfied.

Terminology:

- It is common to say constraint satisfaction problem (CSP) instead of constraint network.
- Strictly speaking, however, "CSP" is the algorithmic problem of finding solutions to constraint networks.

Constraint Networks: Formal

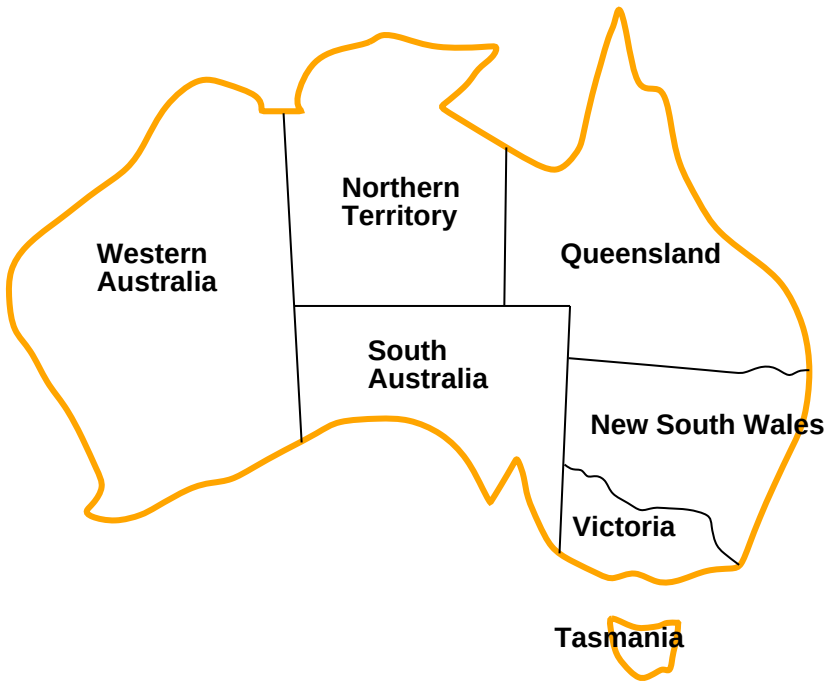
Definition (Constraint Network). *A (binary) constraint network is a triple* $\gamma = (V, D, C)$ *where:*

- $V = \{v_1, \ldots, v_n\}$ *is a finite set of variables.*
- $D = \{D_{v_1}, \ldots, D_{v_n}\}$ *is a corresponding set of finite domains.*
- $C = \{C_{\{u,v\}}\}$ *is a set of binary relations (constraints), where for each* $C_{\{u,v\}}$ *we have* $u, v \in V$, $u \neq v$, *and* $C_{\{u,v\}} \subseteq D_u \times D_v$.
  We require that $C_{\{u,v\}}, C_{\{x,y\}} \in C \implies \{u, v\} \neq \{x, y\}$. *We will write* C_{uv} *instead of* $C_{\{u,v\}}$ *for brevity.*

Notes:

- C_{uv} = permissible combined assignments to u and v.
- Relations are the maximally general formalization of constraints. In illustrations, we often use abbreviations, e.g. "$u \neq v$" etc.
- There is no point in having two constraints C_{uv} and C'_{uv} constrain the same variables u and v, because we can replace them by $C_{uv} \cap C'_{uv}$.
- C_{uv} is identified by its *set* $\{u, v\}$ of variables; the order we choose for the relation is arbitrary.

Example: Coloring Australia



- **Variables:** $V = \{WA, NT, SA, Q, NSW, V, T\}$.
- **Domains:** For all $v \in V$: $D_v = \{red, green, blue\} =: D$.

  $\rightarrow$ If all variables have the same domain, abusing notation we will write D to denote that "global" domain.
- **Constraints:** C_{uv} for adjacent states u and v, with $C_{uv} =$"$u \neq v$", i.e., $C_{uv} = \{(d, d') \in D \times D \mid d \neq d'\}$.

Constraint Networks: Variants

Extensions:

- Infinite domains. (E.g., $D_v = \mathbb{R}$ in Linear Programming.)
- Constraints of higher arity, i.e., relations over $k > 2$ variables. (E.g., propositional CNF satisfiability $\rightarrow$ **Chapter 4**.)

Unary Constraints:

- A unary constraint is a relation C_v over a single variable, i.e., a subset $C_v \subseteq D_v$ of that variable's domain.
- A unary constraint C_v is equivalent to reducing the variable domain, setting $D_v := C_v$.
- Unary constraints are not needed at the formal level. They are often convenient for modeling, i.e., to state "exceptions". (E.g., Australia: D global as on previous slide, but $SA \neq green$.)

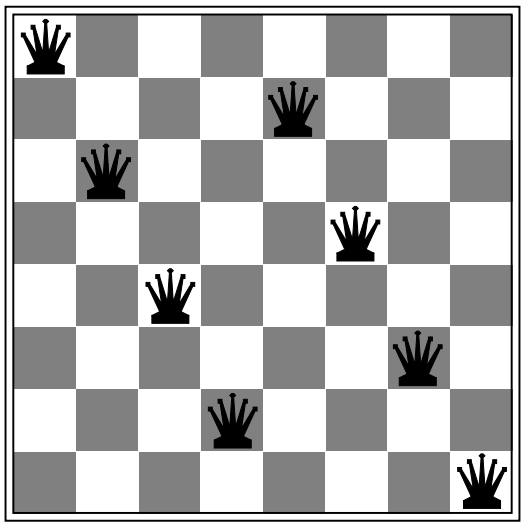
Example: SuDoKu

2	5			3		9		1
	1				4			
4		7				2		8
		5	2					
				9	8	1		
	4				3			
			3	6			7	2
	7							3
9		3				6		4

- **Variables:** $V = \{v_{ij} \mid 1 \leq i, j \leq 9\}$: v_{ij} =cell row i column j.
- **Domains:** For all $v \in V$: $D_v = D = \{1, \ldots, 9\}$.
- **Unary Constraints:** $C_{v_{ij}} = \{d\}$ if cell i, j is pre-filled with d.
- **Binary Constraints:** $C_{v_{ij}v_{i'j'}} =$"$v_{ij} \neq v_{i'j'}$", i.e., $C_{v_{ij}v_{i'j'}} = \{(d, d') \in D \times D \mid d \neq d'\}$, for: $i = i'$ (same row), or $j = j'$ (same column), or $(\lceil \frac{i}{3} \rceil, \lceil \frac{j}{3} \rceil) = (\lceil \frac{i'}{3} \rceil, \lceil \frac{j'}{3} \rceil)$ (same block).

Questionnaire

→ Problem: Place 8 queens so that they don't attack each other.



Question!

How to encode this into a constraint network? Variables? Domains? Constraints?

→ E.g.: Variables: $V = \{v_1, \ldots, v_8\}$: v_i =row of queen in i-th column.
Domains: $D_v = D = \{1, \ldots, 8\}$. Constraints: For $1 \leq i < j \leq 8$:
$C_{v_i v_j} = \{(d, d') \in D \times D \mid d \neq d' \text{ and } |d - d'| \neq |i - j|\}$.

CSP and the Model-and-Solve Paradigm

(some new constraint-reasoning problem)

describe problem as a constraint network $\mapsto$ use off-the-shelf CSP solver

(its solution)

- Constraint networks=generic language to describe this kind of problem.
- CSP solvers=generic algorithms solving such problems.
- The next time you play SuDoKu, just write the game down in CSP format and use an off-the-shelf solver.
- On one of the practical exercise sheets, this is the kind of thing you will be doing ...

Assignments and Consistency

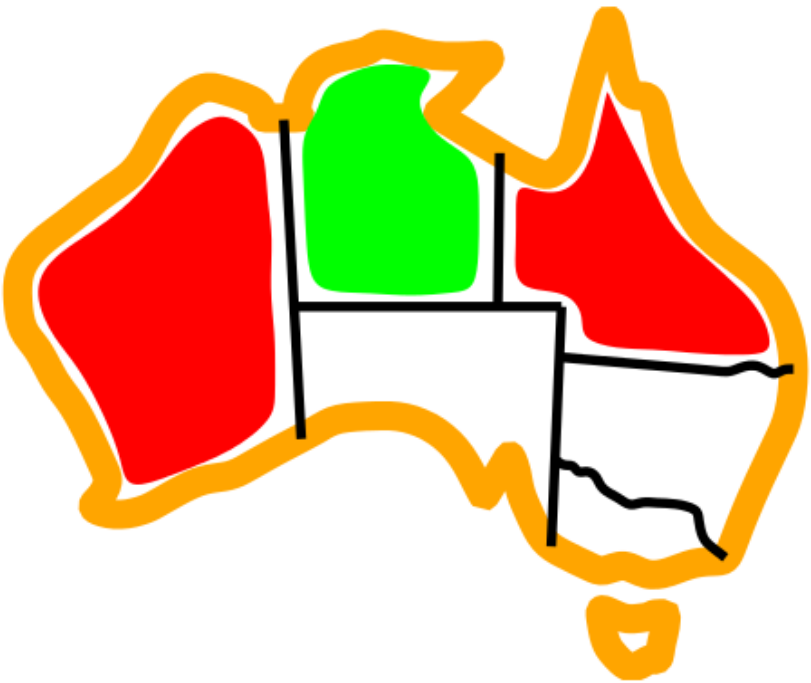
Definition (Assignment). *Let $\gamma = (V, D, C)$ be a constraint network. A partial assignment is a function $a : V' \mapsto \bigcup_{v \in V} D_v$ where $V' \subseteq V$ and $a(v) \in D_v$ for all $v \in V'$. If $V' = V$, then a is a total assignment, or assignment in short.*

$\rightarrow$ A partial assignment assigns some variables to values from their respective domains. A total assignment is defined on all variables.

Definition (Consistency). *Let $\gamma = (V, D, C)$ be a constraint network, and let a be a partial assignment. We say that a is inconsistent if there exist variables $u, v \in V$ on which a is defined, with $C_{uv} \in C$ and $(a(u), a(v)) \notin C_{uv}$. In that case, a violates the constraint C_{uv}. We say that a is consistent if it is not inconsistent.*

$\rightarrow$ Partial assignment inconsistent $=$ "already violates a constraint". (Trivially consistent: The empty assignment.)

Example: Coloring Australia



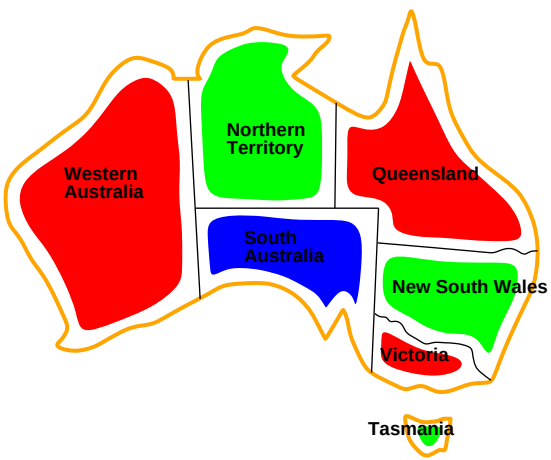
Is this partial assignment consistent? Yes.

Is this partial assignment consistent? No.

Solutions

Definition (Solution). *Let $\gamma = (V, D, C)$ be a constraint network. If a is a total consistent assignment, then a is a solution for γ. If a solution to γ exists, then γ is solvable; otherwise, γ is inconsistent.*

Example "Coloring Australia":



- **Variables:** $V = \{WA, NT, SA, Q, NSW, V, T\}$.
- **Domains:** All $v \in V$: $D_v = D = \{red, green, blue\}$.
- **Constraints:** C_{uv} for adjacent states u and v, with $C_{uv} = \{(d, d') \in D \times D \mid d \neq d'\}$.
- **Solution:** $\{WA = red, NT = green, SA = blue, Q = red, NSW = green, V = red, T = green\}$.

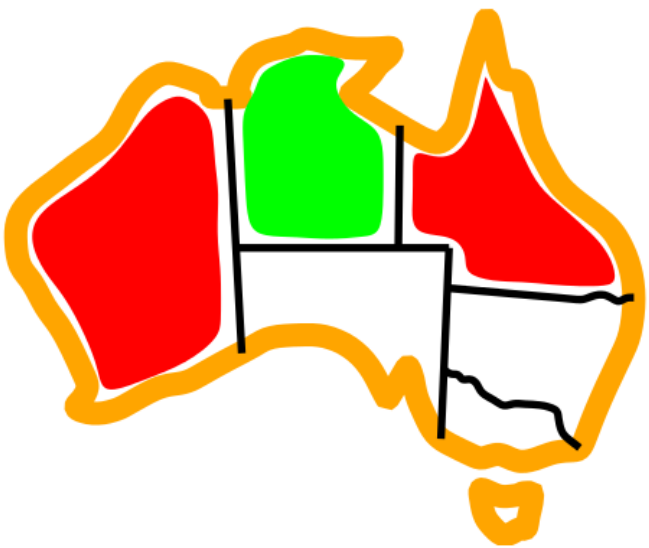
$\rightarrow$ Note: This is not the only solution. E.g., we can permute the colors, and Tasmania can be assigned an arbitrary color.

Consistency vs. Extensibility

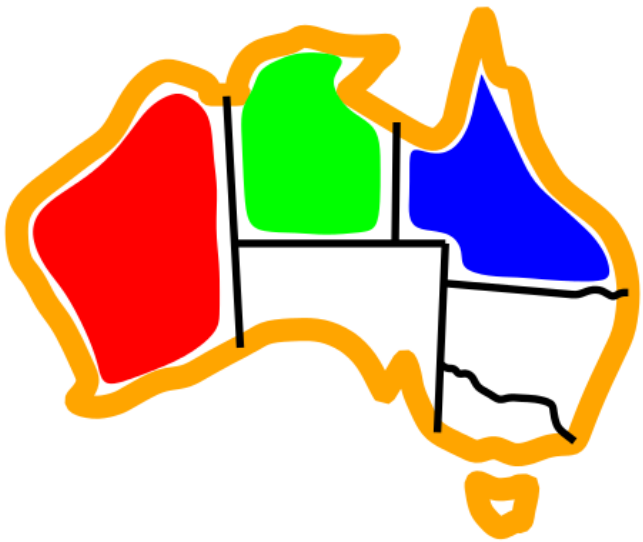
Definition. *Let $\gamma = (V, D, C)$ be a constraint network, and let a be a partial assignment. We say that a can be extended to a solution if there exists a solution a' that agrees with a on the variables where a is defined.*

$\rightarrow$ a can be extended to a solution $\implies$ a consistent. But not vice versa:

Example "Coloring Australia":



Can this partial assignment be extended to a solution? Yes.



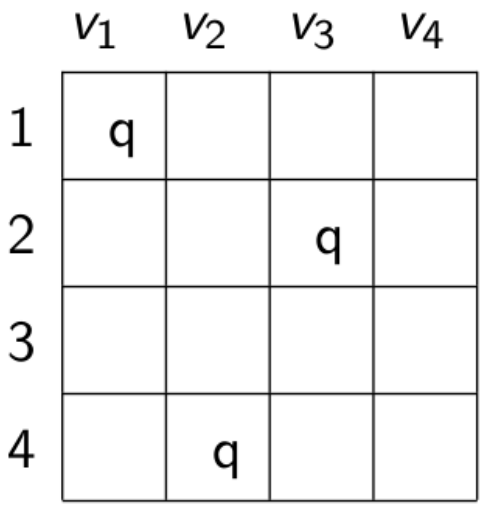
Can this partial assignment be extended to a solution? No.

Consistency vs. Extensibility

Definition. *Let $\gamma = (V, D, C)$ be a constraint network, and let a be a partial assignment. We say that a can be extended to a solution if there exists a solution a' that agrees with a on the variables where a is defined.*

$\rightarrow$ a can be extended to a solution $\implies$ a consistent. But not vice versa:

Example "4-Queens":



Can this partial assignment be extended to a solution? No.

Questionnaire

Question!

Which of the following statements imply that the empty assignment, a_0, can be extended to a solution?

(A): a_0 is consistent.

(B): The network is inconsistent.

(C): There are no binary constraints.

(D): The network is solvable.

$\rightarrow$ (A): No. Being consistent does not imply being extensible to a solution (cf. previous slide). For a_0 in particular: a_0 is always consistent; it can be extended to a solution if and only if the network is solvable.

$\rightarrow$ (B): No. If the network is inconsistent then there are no solutions, so no assignment can be extended to a solution, in particular not a_0.

$\rightarrow$ (C): If one of the unary constraints (variable domains) is empty, then the network is inconsistent and we are in case (B). Otherwise, the network is solvable and a_0 is extensible to a solution.

$\rightarrow$ (D): Yes. The empty assignment can be extended to any solution for the network, if such a solution does exist.

Computational Complexity of CSP

Input size vs. solution space size: Assume constraint network γ with n variables, all with domain size k.

- Number of total assignments: k^n.
- Size of description of γ: nk for variables and domains; at most n^2 constraints, each of size at most $k^2 \implies O(n^2k^2)$.

$\rightarrow$ The number of assignments is exponentially bigger than the size of γ.

It is therefore no surprise that:

Theorem (CSP is NP-complete). *It is* **NP***-complete to decide whether or not a given constraint network γ is solvable.*

Proof. Membership in **NP**: Just guess a total assignment a and verify (in polynomial time) whether a is a solution.

NP-Hardness: The special case of graph coloring (our illustrative example) is known to be **NP**-hard.

Questionnaire

	5	8	7		6	9	4	1
		9	8	1	4	3	5	7
4		7	9		5	2	6	8
3	9	5	2	7	1	4	8	6
7	6	2	4	9	8	1	3	5
8	4	1	6	5	3	7	2	9
1	8	4	3	6	9	5	7	2
5	7	6	1	4	2	8	9	3
9	2	3	5	8	7	6	1	4

Can this partial assignment be extended to a solution? No.

$\rightarrow$ The open cells in the 2nd column can only be filled by 1 and 3. Neither of these fits into the 2nd row (v_{22}).

	5	8	7		6	9	4	1
		9	8		4	3	5	7
4		7	9		5	2	6	8
3	9	5	2	7	1	4	8	6
7	6	2	4	9	8	1	3	5
8	4	1	6	5	3	7	2	9
1	8	4	3	6	9	5	7	2
5	7	6	1	4	2	8	9	3
9	2	3	5	8	7	6	1	4

Can this partial assignment be extended to a solution? Yes.

$\rightarrow$ $v_{22} = 1$, $v_{32} = 3$, $v_{35} = 1$, $v_{25} = 2$, $v_{15} = 3$, $v_{11} = 2$, $v_{21} = 6$. (Compare slide 6.)

Before We Begin

Basic Concepts

- Search: Depth-first enumeration of partial assignments.
- Backtracking: Backtrack at inconsistent partial assignments.
- Inference: Deducing tighter equivalent constraints to reduce search space (backtracking will occur earlier on).

Up next: Naïve backtracking, no inference.

Next Chapter: Backtracking with inference.

Naïve Backtracking

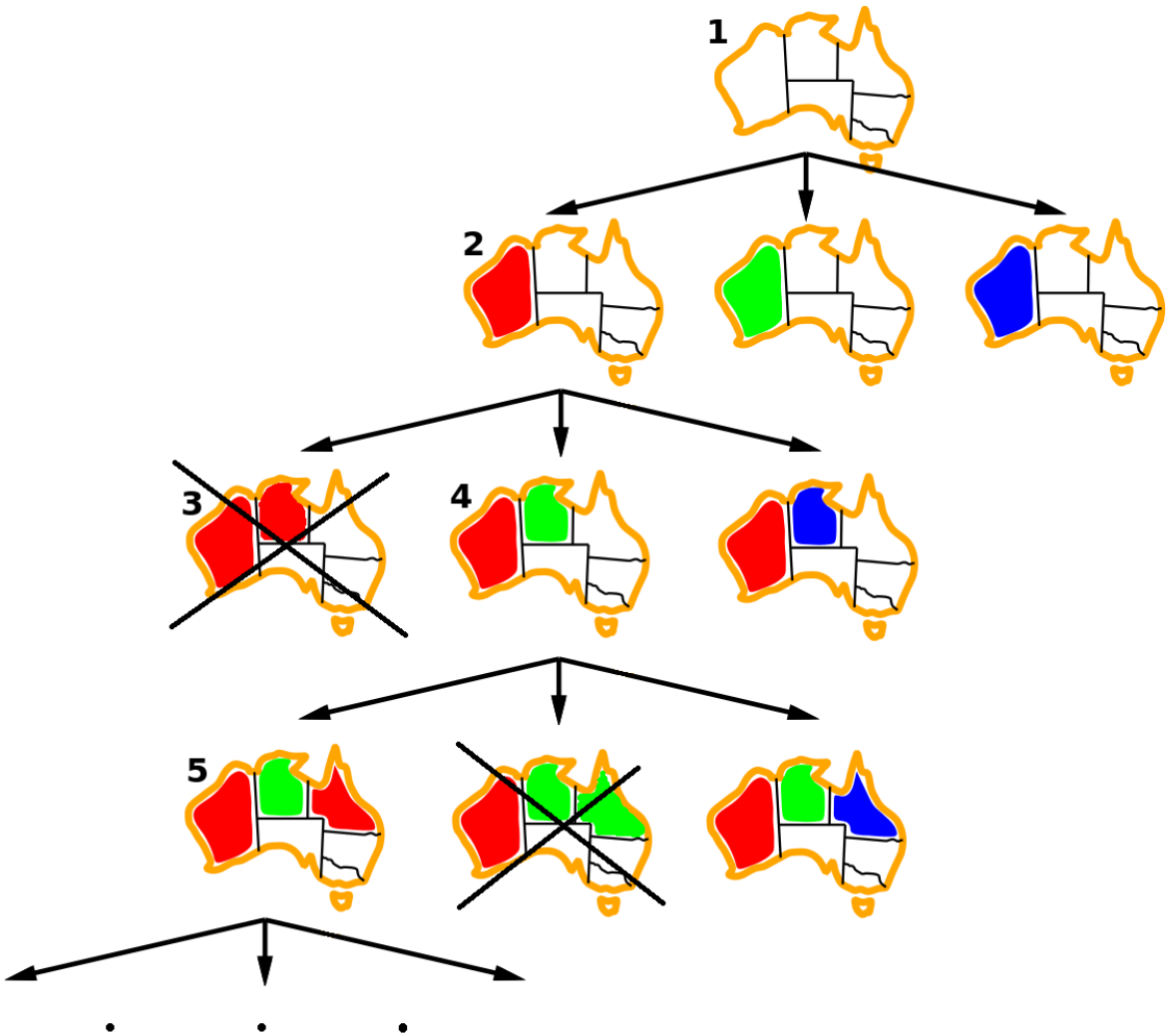
Call with input constraint network γ and the empty assignment a:

function NaïveBacktracking(a) **returns** a solution, or "inconsistent"
  if a is inconsistent **then return** "inconsistent"
  if a is a total assignment **then return** a
  select some variable v for which a is not defined
  for each $d \in D_v$ in some order **do**
    $a' := a \cup \{v = d\}$
    $a'' :=$ NaïveBacktracking(a')
    if $a'' \neq$ "inconsistent" **then return** a''
  return "inconsistent"

→ Backtracking=Recursively instantiate variables one-by-one, backing up out of a search branch if the current partial assignment is already inconsistent.

→ Why is this better than enumerating, and solution-checking, all total assignments (cf. slide 9)? If a partial assignment is already inconsistent, then backtracking does not enumerate any extensions thereof.

Example: Coloring Australia



Naïve Backtracking, Pro and Contra

Pro:

- Naïve backtracking is extremely simple. (You can implement it on a Commodore 128.)
- Despite this simplicity, it is *much* more efficient than enumerating total assignments. (You can implement it on a Commodore 128 *and* solve the Bundesliga.)
- Naïve backtracking is complete (if there is a solution, backtracking will find it).

Contra:

- Backtracking does not recognize a that cannot be extended to a solution, unless a is already inconsistent.
  $\rightarrow$ Employ inference to improve this! (**Chapter 8**).

Naïve backtracking, Pro and Contra: Illustration

	5	8	7		6	9	4	1
		9	8	1	4	3	5	7
4		7	9		5	2	6	8
3	9	5	2	7	1	4	8	6
7	6	2	4	9	8	1	3	5
8	4	1	6	5	3	7	2	9
1	8	4	3	6	9	5	7	2
5	7	6	1	4	2	8	9	3
9	2	3	5	8	7	6	1	4

Much more efficient than enumerating total assignments: "fill cells one-by-one and stop already when an illegal row/column/block occurs" vs. "fill all cells, then check whether it's a solution".

Does not recognize a that cannot be extended to a solution, unless a is already inconsistent: "don't think at all, just fill in cells and see where you get to". In the present example, we would not even try to see the issue, and instead just keep filling in values, e.g. trying to put 5 or 8 into the top left cell.

$\rightarrow$ "Human SuDoKu playing" $=$ lots of inference!
(You want to minimize the number of failed attempts to keep track of on paper . . .)

Questionnaire

Question!

Say G is a clique of n vertices, and we run backtracking for graph coloring with n different colors. How big is the search space (consistent partial assignments) of naïve backtracking?

(A): n (B): $n!$

(C): $1 + \sum_{i=0}^{n-1} n * \cdots * (n-i)$ (D): n^n

$\rightarrow$ (C): 1 for the root. At the first vertex, we have n consistent colors, then $n-1$ consistent colors, etc. We need to add up the nodes at each layer of the search tree.

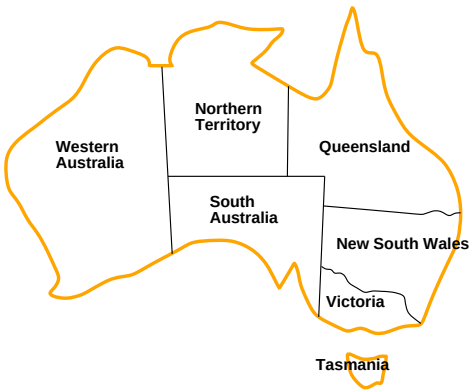
Question!

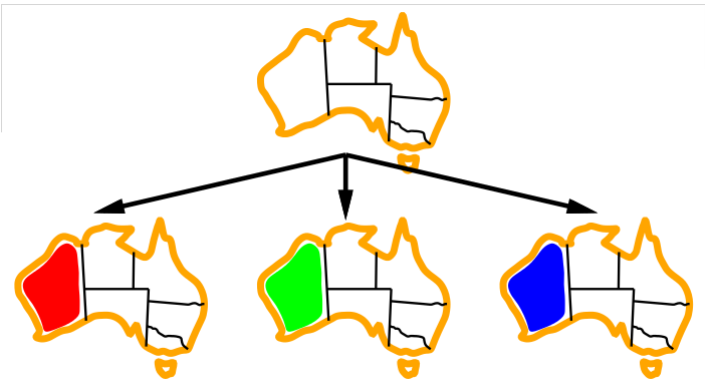
If G is a line and we order variables left-to-right?

(A): $1 + \sum_{i=0}^{n-1} n * \cdots * (n-i)$ (B): $1 + \sum_{i=0}^{n-1} n * (n-1)^i$

$\rightarrow$ (B): At the first vertex, we have n consistent colors; at each later vertex it's $n-1$.

Questionnaire, ctd.





Question!

Variable order WA, NT, Q, NSW, V, T, SA. Tightest upper bound on naïve backtracking search space size?

(A): 145 (B): 382

(C): 433 (D): 3^7

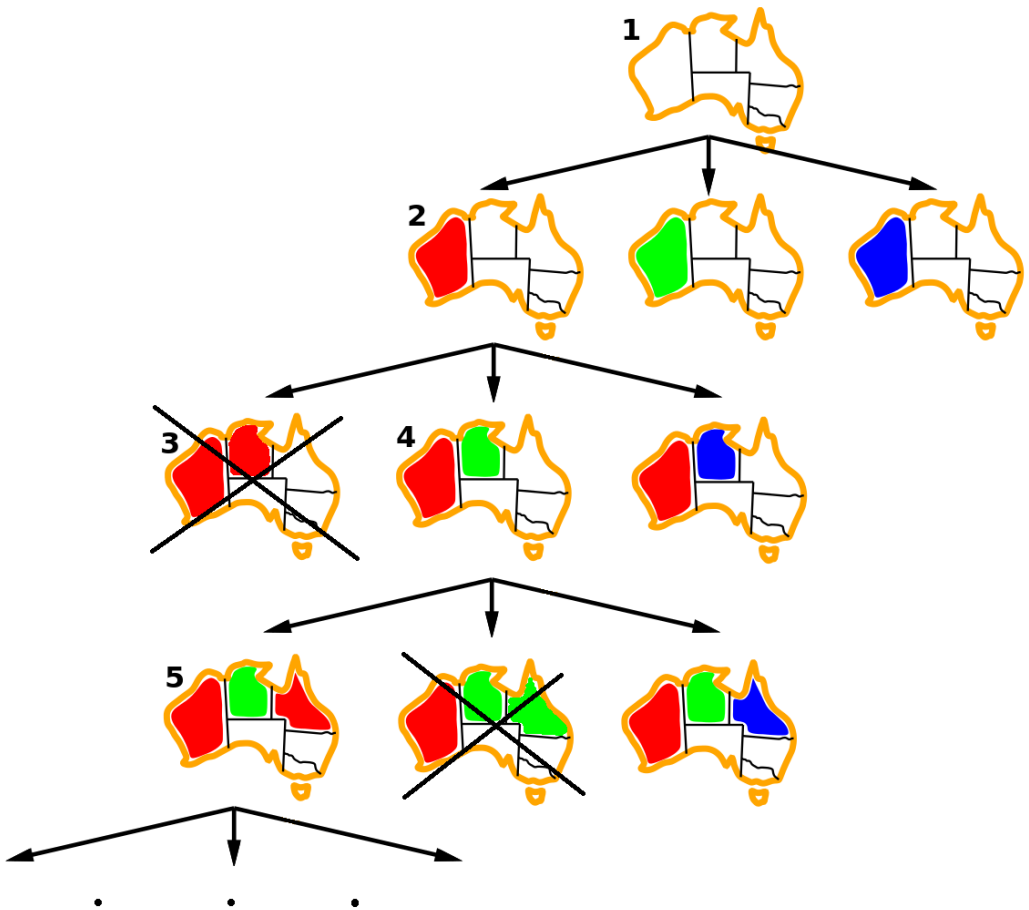
$\rightarrow$ (B): With this variable order, we have: 3 choices for WA (3 nodes); 2 (consistent!) choices for NT ($2 * 3 = 6$ new nodes); 2 choices each for Q, NSW, V ($12, 24, 48$ new nodes); 3 choices for T (144 new nodes). For each of the 144 leaves, we have either 0 or 1 choices for SA. To get an upper bound, we conservatively assume it's 1 everywhere, adding another 144 consistent nodes. Plus 1 for the root.

What to Order, Where, in Naïve Backtracking

function NaïveBacktracking(γ, a) **returns** a solution, or "inconsistent"
  if a is inconsistent with γ **then return** "inconsistent"
  if a is a total assignment **then return** a
  select some variable v for which a is not defined
  for each $d \in D_v$ in some order **do**
    $a' := a \cup \{v = d\}$
    $a'' :=$ NaïveBacktracking(γ, a')
    if $a'' \neq$ "inconsistent" **then return** a''
  return "inconsistent"

$\rightarrow$ The order in which we consider variables and their values may have a huge impact on search space size!

Example: Coloring Australia



- WA, NT, Q as on slide 32 $\Longrightarrow$ $3 * 2 * 2$.
- Any ideas for better variable orders? For SA, WA, NT it's $3 * 2 * 1$.
  $\rightarrow$ The "most important/most restricted variables" first.

Variable- and Value Ordering

Variable Ordering:

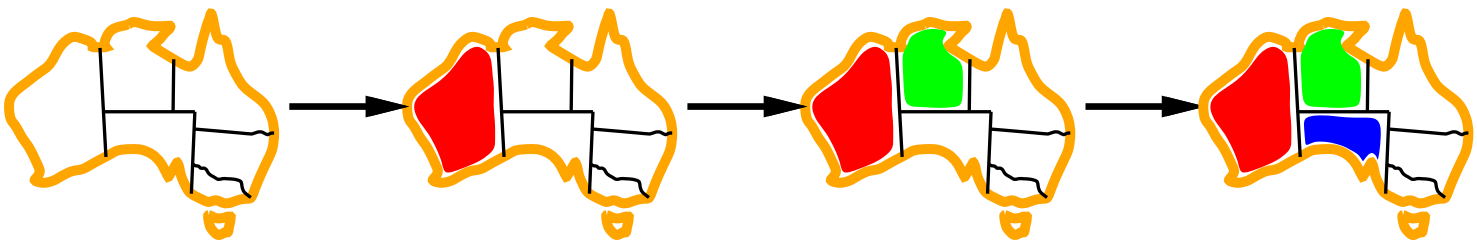
- Naïve backtracking does not specify in which order the variables are considered.
- That ordering often dramatically influences search space size. (Cf. previous slide, and slide 36 vs. slide 44.)

Value Ordering:

- Naïve backtracking does not specify in which order the values of the chosen variable are considered.
- If no solution exists below current node: Doesn't matter, we will have to search the whole sub-tree anyway.
- If solution does exist below current node: Does matter. If we always chose a "correct" value (from a solution) then no backtracking is needed.

Variable Ordering Strategy, Part I

A commonly used strategy: most constrained variable first. Always pick a variable v with minimal $|\{d \in D_v \mid a \cup \{v = d\} \text{ is consistent}\}|$.

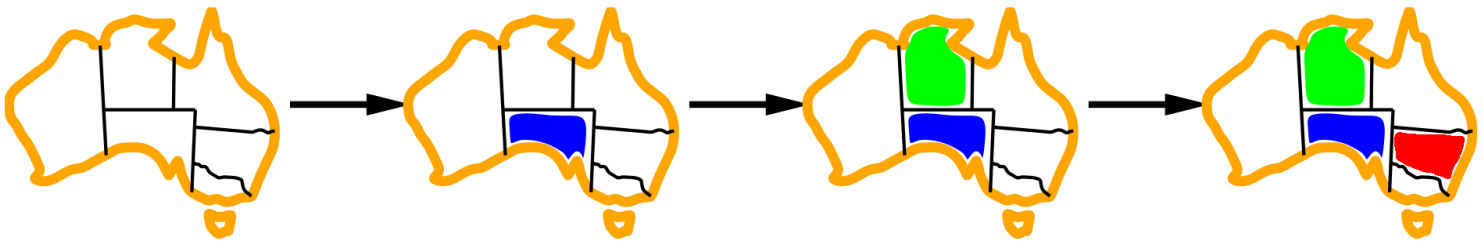


$\rightarrow$ By choosing a most constrained variable v first, we reduce the branching factor (number of sub-trees generated for v) and thus reduce the size of our search tree.

$\rightarrow$ Extreme case: If $|\{d \in D_v \mid a \cup \{v = d\} \text{ is consistent}\}| = 1$, then the value assignment to v is forced by our previous choices.

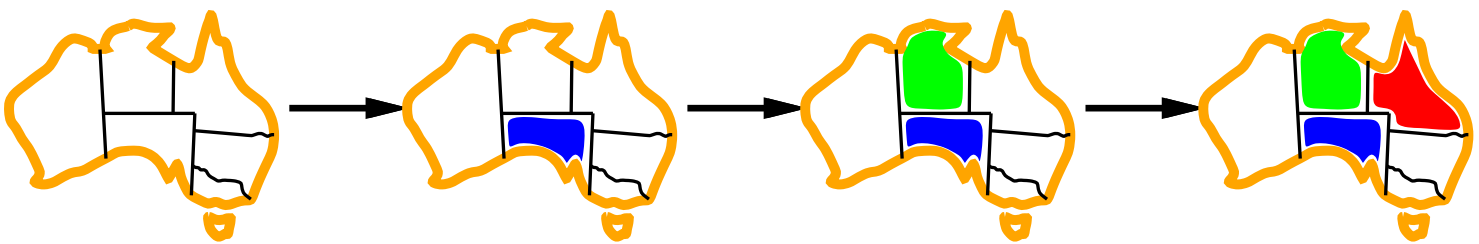
Variable Ordering Strategy, Part II

Another commonly used strategy: most constraining variable first.
Always pick v with maximal $|\{u \in V \mid a(u) \text{ is undefined}, C_{uv} \in C\}|$.



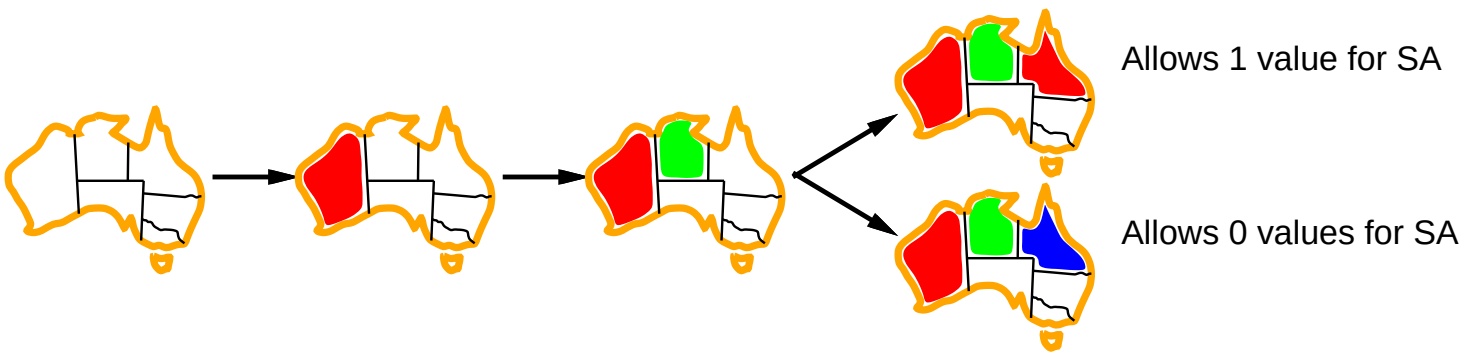
$\rightarrow$ By choosing a most constraining variable first, we detect inconsistencies earlier on and thus reduce the size of our search tree.

Commonly used strategy combination: From the set of most constrained variables, pick a most constraining variable.



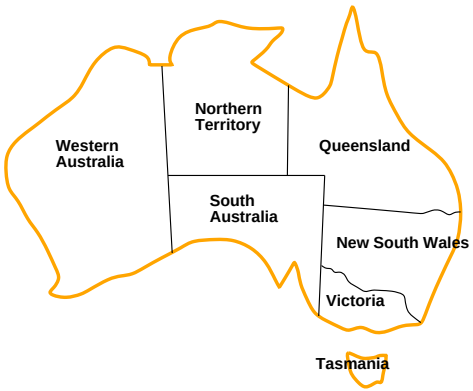
Value Ordering Strategy

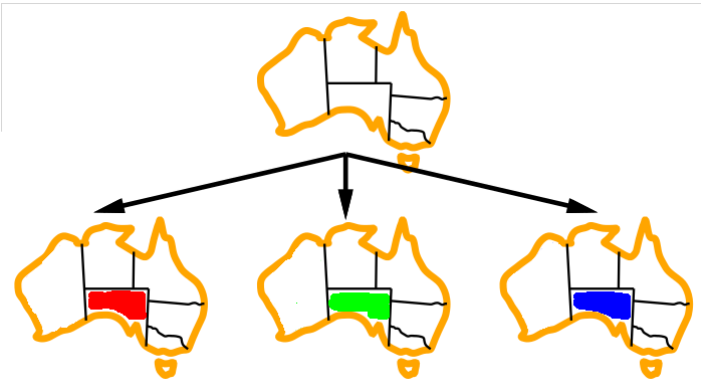
A commonly used strategy: least constraining value first. For variable v, always pick $d \in D_v$ with
minimal $|\{d' \mid d' \in D_u, a(u) \text{ is undefined}, C_{uv} \in C, (d', d) \notin C_{uv}\}|$.



→ By choosing a least constraining value first, we increase the chances to not rule out the solutions below the current node.

Questionnaire





Question!

Variable order SA, NT, Q, NSW, V, WA, T. Tightest upper bound on naïve backtracking search space size?

(A): 52 (B): 145

(C): 382 (D): 433

$\rightarrow$ (A): With this variable order, we have 3 choices for SA and 2 choices for NT, yielding $1 + 3 + 3 * 2 = 10$ search nodes and $3 * 2 = 6$ tree leaves. For each of Q, NSW, V, WA, we have only 1 choice, so each adds another layer of 6 nodes. We have 3 choices for T, adding another $3 * 6 = 18$ nodes. This sums up to 52.

$\rightarrow$ This is the strategy combination from slide 42. Compare with slide 36!

Summary

- Constraint networks γ consist of variables, associated with finite domains, and constraints which are binary relations specifying permissible value pairs.
- A partial assignment a maps some variables to values, a total assignment does so for all variables. a is consistent if it complies with all constraints. A consistent total assignment is a solution.
- The constraint satisfaction problem (CSP) consists in finding a solution for a constraint network. This has numerous applications including, e.g., scheduling and timetabling.
- Backtracking instantiates variables one-by-one, pruning inconsistent partial assignments.
- Variable orderings in backtracking can dramatically reduce the size of the search tree. Value orderings have this potential (only) in solvable sub-trees.

→ **Next Chapter**: Inference and decomposition, for improved efficiency.

Reading

- *Chapter 6: Constraint Satisfaction Problems*, Sections 6.1 and 6.3 [Russell and Norvig (2010)].

  Content: Compared to our treatment of the topic "Constraint Satisfaction Problems" (Chapters 9 and 10), RN covers much more material, but less formally and in much less detail (in particular, my slides contain many additional in-depth examples). Nice background/additional reading, can't replace the lecture.

  Section 6.1: Similar to my "Introduction" and "Constraint Networks", less/different examples, much less detail, more discussion of extensions/variations.

  Section 6.3: Similar to my "Naïve Backtracking" and "Variable- and Value Ordering", with less examples and details; contains part of what I cover in Chapter 10 (RN does inference first, then backtracking). Additional discussion of *backjumping*.

References I

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (Third Edition)*. Prentice-Hall, Englewood Cliffs, NJ, 2010.