Artificial Intelligence

CSP 1: Constraint Satisfaction Problems

Jendrik Seipp

Linköping University

based on slides by Thomas Keller and Malte Helmert (University of Basel)

Questions?

post feedback and ask questions anonymously at

`https://padlet.com/jendrikseipp/tddc17`

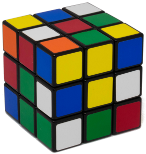
Intended Learning Outcomes

- explain what “constraint satisfaction problems” (CSPs) are
- model and solve simple CSPs

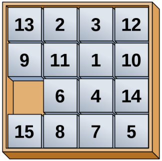
Constraint Satisfaction Problems

Constraint Satisfaction Problems

(heuristic) search algorithms considered so far:

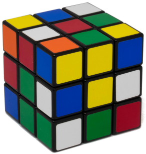


- wide variety of problems
- problem-specific heuristics
- no general solver

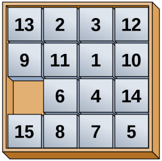


Constraint Satisfaction Problems

(heuristic) search algorithms considered so far:

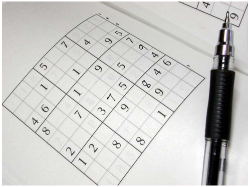


- wide variety of problems
- problem-specific heuristics
- no general solver

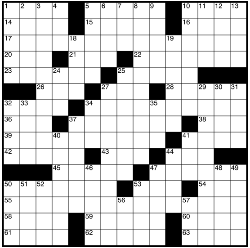


constraint satisfaction problems (CSP) considered today:

- problem scope more restricted
- problem-inpdendent methods
- general solver

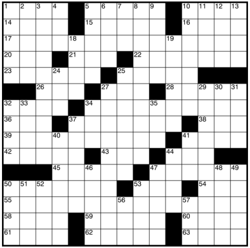


Informal Description



- a constraint is a condition that every solution to a problem must satisfy
- a CSP is defined by
  - a finite set of variables
  - a domain for each variable
  - a set of constraints
- a solution for a CSP is an assignment of each variable to a value in its domain that violates no constraint

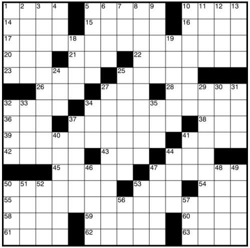
Informal Description



- a constraint is a condition that every solution to a problem must satisfy
- a CSP is defined by
  - a finite set of variables
  - a domain for each variable
  - a set of constraints
- a solution for a CSP is an assignment of each variable to a value in its domain that violates no constraint

- variables and domains in cross-word puzzle? constraints?

Informal Description

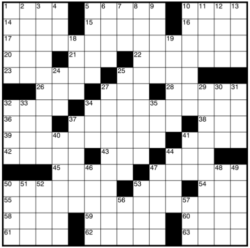


- a constraint is a condition that every solution to a problem must satisfy
- a CSP is defined by
  - a finite set of variables
  - a domain for each variable
  - a set of constraints
- a solution for a CSP is an assignment of each variable to a value in its domain that violates no constraint

- variables and domains in cross-word puzzle? constraints?

$\longrightarrow$ one variable for each cell with domain $\{A, \ldots, Z\}$

Informal Description



- a constraint is a condition that every solution to a problem must satisfy
- a CSP is defined by
  - a finite set of variables
  - a domain for each variable
  - a set of constraints
- a solution for a CSP is an assignment of each variable to a value in its domain that violates no constraint

- variables and domains in cross-word puzzle? constraints?

$\longrightarrow$ one variable for each cell with domain $\{A, \ldots, Z\}$

$\longrightarrow$ constraints: length of “boxes”, horizontal and vertical words must not contradict each other

Definition

Definition (Constraint Satisfaction Problem)

A constraint satistfaction problem (or constraint network) is a 3-tuple $\mathcal{C} = \langle V, dom, C \rangle$ such that:

- V is a non-empty and finite set of variables,
- dom is a function that assigns a non-empty domain to each variable $v \in V$, and
- C is a set of constraints $c = \langle \text{scope}, \text{rel} \rangle$, where $\text{scope}(c) = \langle v_1, \ldots, v_n \rangle$ is a n-tuple of (pairwise distinct) variables and $\text{rel}(c) \subseteq dom(v_1) \times \cdots \times dom(v_n)$

restrictions considered here:

- finite domains
- unary or binary constraints

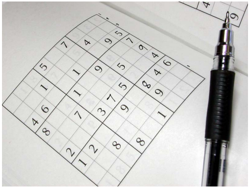
Unary Constraints

- a unary constraint c has scope$(c) = (v)$ for $v \in V$
- c restricts $dom(v)$ to the values allowed by c
- it is often useful to have additional restrictions on single variables as constraints
- formally, unary constraints are not necessary, but they often allow to describe CSPs more clearly

Unary Constraints

- a unary constraint c has $\mathrm{scope}(c) = (v)$ for $v \in V$
- c restricts $dom(v)$ to the values allowed by c
- it is often useful to have additional restrictions on single variables as constraints
- formally, unary constraints are not necessary, but they often allow to describe CSPs more clearly

- $dom(v) = \{1, \ldots, 9\}$ for all $v \in V$ (for all Sudoku instances)
- $c = \langle (v_{13}), \{(4)\} \rangle$ (for a specific instance)



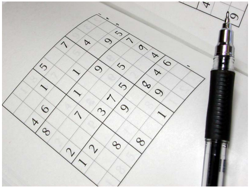
Binary Constraints

- a binary constraint has $\text{scope}(c) = (u, v)$ for $u, v \in V$, $u \neq v$
- c expresses which joint assignments to u and v are allowed
- c is trivial if $\text{rel}(c) = dom(u) \times dom(v)$
  $\rightsquigarrow$ c is usually not given explicitly (c exists formally)
- constraint c' with $\text{scope}(c') = (v, u)$ refers to same variables $\rightsquigarrow$ only c or c' is usually given explicitly (both exist formally)

Binary Constraints

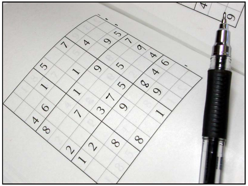
- a binary constraint has $\text{scope}(c) = (u, v)$ for $u, v \in V$, $u \neq v$
- c expresses which joint assignments to u and v are allowed
- c is trivial if $\text{rel}(c) = dom(u) \times dom(v)$
  $\rightsquigarrow$ c is usually not given explicitly (c exists formally)
- constraint c' with $\text{scope}(c') = (v, u)$ refers to same variables $\rightsquigarrow$ only c or c' is usually given explicitly (both exist formally)

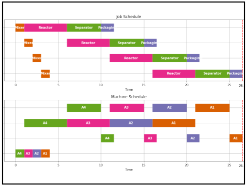
$$c = \langle (v_{11}, v_{21}), \{(x, y) \mid x \neq y\} \rangle$$
$$c' = \langle (v_{21}, v_{11}), \{(y, x) \mid y \neq x\} \rangle$$

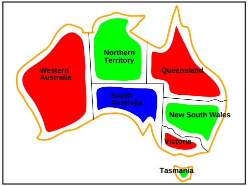


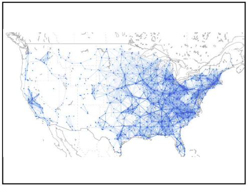
Examples

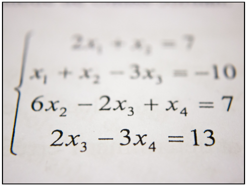
CSP Examples

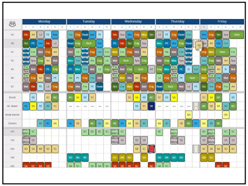








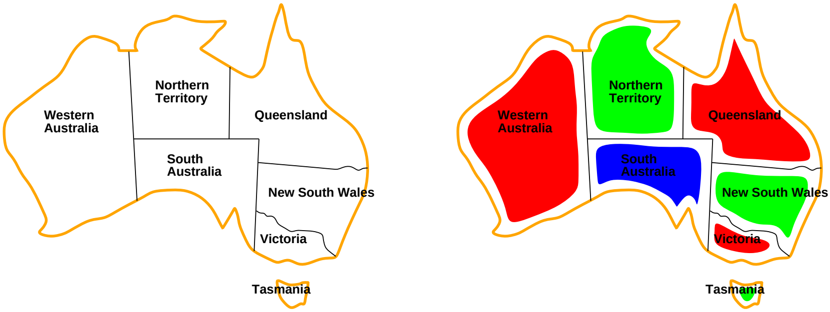




Example: Graph Coloring

Given a graph and $k \in \mathbb{N}$, how can we

- color the vertices using k colors
- such that two neighboring vertices never have the same color?

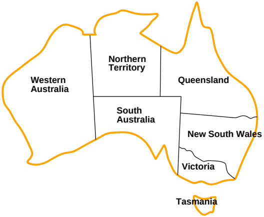


How to formalize this CSP?

Example: Graph Coloring

Given a graph and $k \in \mathbb{N}$, how can we

- color the vertices using k colors
- such that two neighboring vertices never have the same color?



- variables: $V = \{WA, NT, Q, NSW, VI, SA, T\}$
- domains: $dom(v) = \{r, g, b\}$ for all $v \in V$
- constraints: $\{c_{uv} \mid \text{for all connected } u, v \in V\}$, where

  $c_{uv} = \langle (u, v), \{(k, \ell) \in \{r, g, b\} \times \{r, g, b\} \mid k \neq \ell\}\rangle$

e.g., $c_{WA,NT} = \langle (WA, NT), \{(r, g), (r, b), (g, r), (g, b), (b, r), (b, g)\}\rangle$

Four Color Problem

famous problem in mathematics: Four Color Problem

- Is it always possible to color a planar graph with 4 colors?
- conjectured by Francis Guthrie (1852)
- 1890 first proof that 5 colors suffice
- several wrong proofs surviving for over 10 years

Four Color Problem

famous problem in mathematics: Four Color Problem

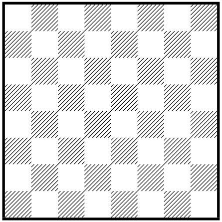
- Is it always possible to color a planar graph with 4 colors?
- conjectured by Francis Guthrie (1852)
- 1890 first proof that 5 colors suffice
- several wrong proofs surviving for over 10 years
- solved by Appel and Haken in 1976: 4 colors suffice
- Appel and Haken reduced the problem to 1936 cases, which were then checked by computers
- first famous mathematical problem solved (partially) by computers
  $\rightsquigarrow$ led to controversy: is this a mathematical proof?

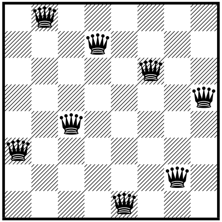
numberphile video:
`https://www.youtube.com/watch?v=NgbK43jB4rQ`

Example: 8 Queens Problem

How can we

- place 8 queens on a chess board
- such that no two queens threaten each other?



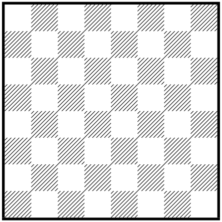


- originally proposed in 1848
- variants: board size; other pieces; higher dimension
- there are 92 solutions (12 non-symmetric ones)

Example: 8 Queens Problem

How can we

- place 8 queens on a chess board
- such that no two queens threaten each other?



- variables: $V = \{v_1, \ldots, v_8\}$
- domains: $dom(v) = \{1, \ldots, 8\}$ for all $v \in V$
- constraints: $\{c_{ij} \mid \text{for all } 1 \leq i < j \leq 8\}$, where

$$c_{i,j} = \langle (v_i, v_j), \{(k, \ell) \in \{1, \ldots, 8\} \times \{1, \ldots, 8\} \mid k \neq \ell \wedge |k - \ell| \neq |i - j|\} \rangle$$

e.g., $$c_{1,3} = \langle (v_1, v_3), \{(1,2), (1,4), (1,5), (1,6), (1,7), (1,8), (2,1), (2,3), (2,5), (2,6), (2,7), (2,8), \ldots, (8,1), (8,2), (8,3), (8,4), (8,5), (8,7)\} \rangle$$

Example: Sudoku

How can we

- completely fill an already partially filled 9×9 matrix with numbers from {1,2,...,9}
- such that each row, each column, and each of the nine 3×3 blocks contains every number exactly once?

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

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

Example: Sudoku

How can we

- completely fill an already partially filled 9 × 9 matrix with numbers from {1,2,...,9}
- such that each row, each column, and each of the nine 3 × 3 blocks contains every number exactly once?

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: $V = \{v_{ij} \mid 1 \leq 1, j \leq 9\}$
- domains: $dom(v) = \{1, \ldots, 9\}$ for all $v \in V$
- unary constraints: $c_{v_{ij}} = \{k\}$ for all cells $\langle i, j \rangle$ with predefined value k
- binary constraints: $c_{v_{ij}v_{i'j'}} = \langle (v_{ij}, v_{i'j'}), \{(a, b) \in \{1, \ldots, 9\}^2 \mid a \neq b\}$ for all $v_{ij}, v_{i'j'} \in V$ with
  - $i = i'$ (same row), or
  - $j = j'$ (same column), or
  - $\langle \lceil \frac{i}{3} \rceil, \lceil \frac{j}{3} \rceil \rangle = \langle \lceil \frac{i'}{3} \rceil, \lceil \frac{j'}{3} \rceil \rangle$ (same block)

Solutions

Assignments

Definition (assignment, partial assignment)

Let $\mathcal{C} = \langle V, dom, C \rangle$ be a CSP.

A partial assignment of $\mathcal{C}$ (or of V) is a function

$$\alpha : V' \to \bigcup_{v \in V} dom(v)$$

with $V' \subseteq V$ and $\alpha(v) \in dom(v)$ for all $v \in V'$.

If $V' = V$, then α is also called total assignment (or assignment).

- $\leadsto$ partial assignments assign values to some or to all variables
- $\leadsto$ (total) assignments are defined on all variables

Consistency

Definition (inconsistent, consistent, violated)

A partial assignment α of a CSP $\mathcal{C}$ is called inconsistent if there are variables u, v such that α is defined for both u and v,
and there is $c \in C$ s.t. $\text{scope}(c) = (u, v)$ and $(\alpha(u), \alpha(v)) \notin \text{rel}(c)$.

In this case, we say α violates the constraint c.

A partial assignment is called consistent if it is not inconsistent.

Solution

Definition (solution, solvable)

Let C be a CSP.

A consistent and total assignment of C is called a solution of C.

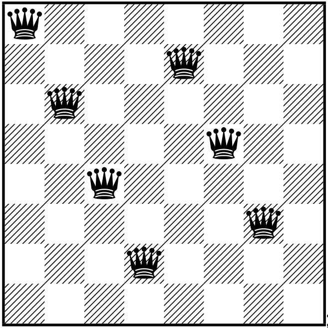
If a solution of C exists, C is called solvable.

If no solution exists, C is called inconsistent.

Consistency vs. Solvability

Note: Consistent partial assignments α cannot necessarily be extended to a solution.

It only means that so far (i.e., on the variables where α is defined) no constraint is violated.


;

$$\alpha = \{v_1 \mapsto 1, v_2 \mapsto 3, v_3 \mapsto 5, v_4 \mapsto 7, v_5 \mapsto 2, v_6 \mapsto 4, v_7 \mapsto 6\}$$

Complexity

Complexity of Constraint Satisfaction Problems

Proposition (CSPs are NP-complete)

Deciding whether a given CSP is solvable is NP-complete.

Proof

Membership in NP:

Guess and check: guess a solution and check it for validity.
This can be done in polynomial time in the size of the input.

NP-hardness:

The graph coloring problem is a special case of CSPs
and is already known to be NP-complete.

Compact Encodings and General CSP Solvers

CSPs allow for compact encodings of large sets of assignments:

- consider a CSP with n variables with domains of size k

$\rightsquigarrow$ k^n assignments

Compact Encodings and General CSP Solvers

CSPs allow for compact encodings of large sets of assignments:

- consider a CSP with n variables with domains of size k

⇝ k^n assignments

- for the description as a CSP, at most $\binom{n}{2}$, i.e., $O(n^2)$ constraints have to be provided
- every (binary) constraint consists of at most $O(k^2)$ pairs

⇝ encoding size $O(n^2 k^2)$

⇝ the number of assignments is exponentially larger than the description of the CSP

Compact Encodings and General CSP Solvers

CSPs allow for compact encodings of large sets of assignments:

- consider a CSP with n variables with domains of size k

$\rightsquigarrow$ k^n assignments

- for the description as a CSP, at most $\binom{n}{2}$, i.e., $O(n^2)$ constraints have to be provided
- every (binary) constraint consists of at most $O(k^2)$ pairs

$\rightsquigarrow$ encoding size $O(n^2k^2)$

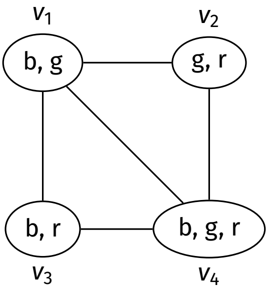
$\rightsquigarrow$ the number of assignments is exponentially larger than the description of the CSP

- as a consequence, such descriptions can be used as inputs of general constraint solvers

Exercise

CSP Exercise

Consider a variant of the graph coloring problem where each vertex has a set of allowed colors.



1. Formalize the example as a binary constraint network.
2. Is the constraint network solvable? If yes, provide a solution of the constraint network. If not, justify your answer.
3. Provide a minimal consistent partial assignment that cannot be extended to a solution.
4. Provide an inconsistent partial assignment.

CSP Exercise – Solution

1. $C = \langle V, dom, (R_{uv}) \rangle$ with
   - $V = \{v_1, \ldots, v_4\}$,
   - $dom(v_1) = \{b, g\}$, $dom(v_2) = \{g, r\}$, $dom(v_3) = \{b, r\}$, $dom(v_4) = \{b, g, r\}$, and
   - (binary) constraints:

$$
\begin{aligned}
R_{v_1,v_2} &= \{\langle b, g\rangle, \langle b, r\rangle, \langle g, r\rangle\} \\
R_{v_1,v_3} &= \{\langle b, r\rangle, \langle g, b\rangle, \langle g, r\rangle\} \\
R_{v_1,v_4} &= \{\langle b, g\rangle, \langle b, r\rangle, \langle g, b\rangle, \langle g, r\rangle\} \\
R_{v_2,v_4} &= \{\langle g, b\rangle, \langle g, r\rangle, \langle r, b\rangle, \langle r, g\rangle\} \\
R_{v_3,v_4} &= \{\langle b, g\rangle, \langle b, r\rangle, \langle r, b\rangle, \langle r, g\rangle\}
\end{aligned}
$$

2. Solvable. Solution: $\alpha_1 = \{v_1 \mapsto b, v_2 \mapsto r, v_3 \mapsto r, v_4 \mapsto g\}$

   (There is second solution: $\alpha_2 = \{v_1 \mapsto g, v_2 \mapsto r, v_3 \mapsto r, v_4 \mapsto b\}$)
3. $\alpha_1 = \{v_2 \mapsto g\}$ or $\alpha_2 = \{v_4 \mapsto r\}$ or $\alpha_3 = \{v_3 \mapsto b\}$
4. Inconsistent partial assignment: $\alpha = \{v_1 \mapsto b, v_3 \mapsto b\}$

Artificial Intelligence

CSP 2: Backtracking and Inference

Jendrik Seipp

Linköping University

based on slides by Thomas Keller and Malte Helmert (University of Basel)

CSP Algorithms

we now consider algorithms for solving CSPs

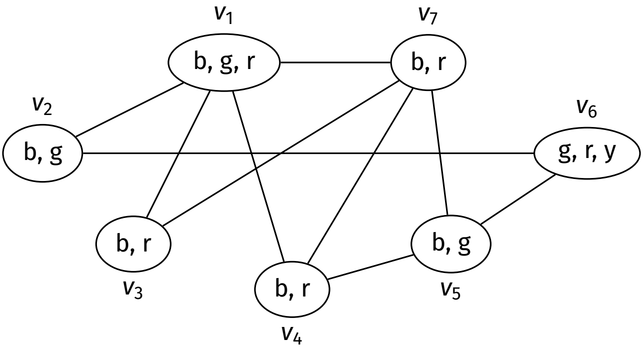
basic concepts:

- search: check partial assignments systematically
- backtracking: discard inconsistent partial assignments
- inference: derive equivalent, but tighter constraints
  to reduce the size of the search space

Backtracking Without Inference (= Naive Backtracking)

Naive Backtracking: Example

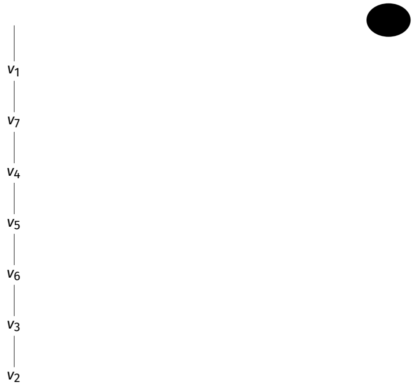
Consider the CSP for the following graph coloring instance:



Naive Backtracking: Example

search tree for naive backtracking with

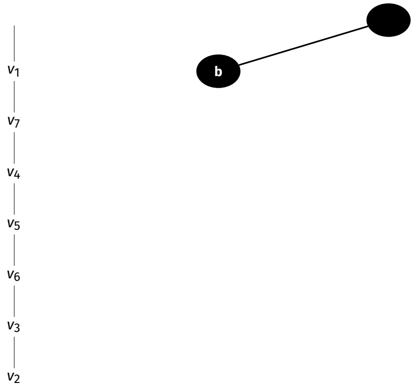
- fixed variable order $v_1, v_7, v_4, v_5, v_6, v_3, v_2$
- alphabetical order of the values



Naive Backtracking: Example

search tree for naive backtracking with

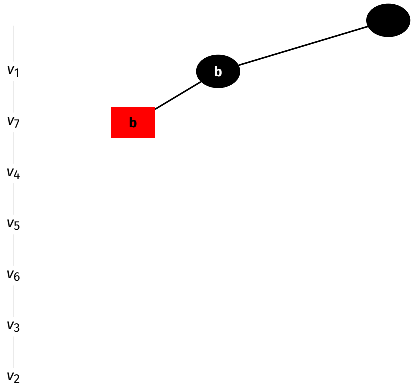
- fixed variable order $v_1, v_7, v_4, v_5, v_6, v_3, v_2$
- alphabetical order of the values



Naive Backtracking: Example

search tree for naive backtracking with

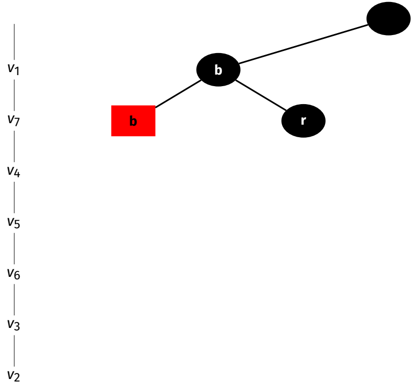
- fixed variable order $v_1, v_7, v_4, v_5, v_6, v_3, v_2$
- alphabetical order of the values



Naive Backtracking: Example

search tree for naive backtracking with

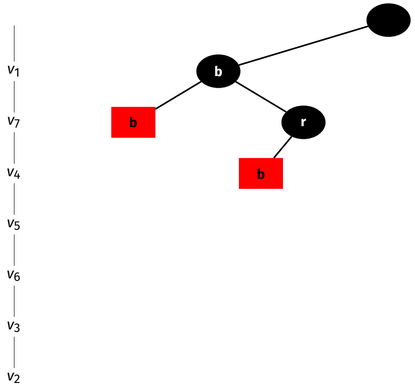
- fixed variable order $v_1, v_7, v_4, v_5, v_6, v_3, v_2$
- alphabetical order of the values



Naive Backtracking: Example

search tree for naive backtracking with

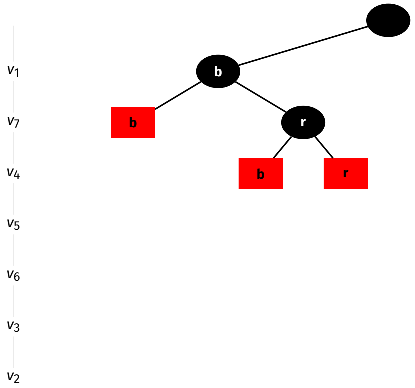
- fixed variable order $v_1, v_7, v_4, v_5, v_6, v_3, v_2$
- alphabetical order of the values



Naive Backtracking: Example

search tree for naive backtracking with

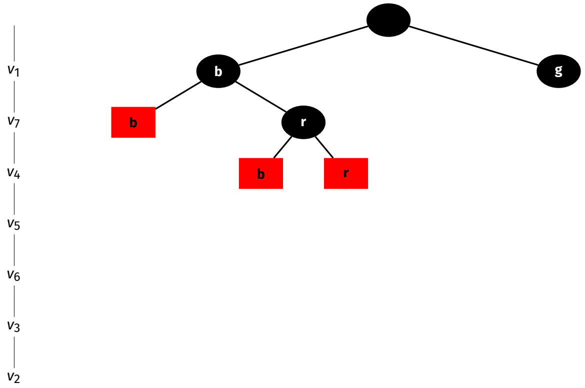
- fixed variable order $v_1, v_7, v_4, v_5, v_6, v_3, v_2$
- alphabetical order of the values



Naive Backtracking: Example

search tree for naive backtracking with

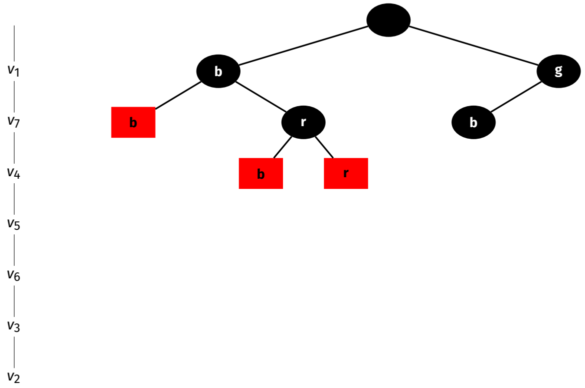
- fixed variable order $v_1, v_7, v_4, v_5, v_6, v_3, v_2$
- alphabetical order of the values



Naive Backtracking: Example

search tree for naive backtracking with

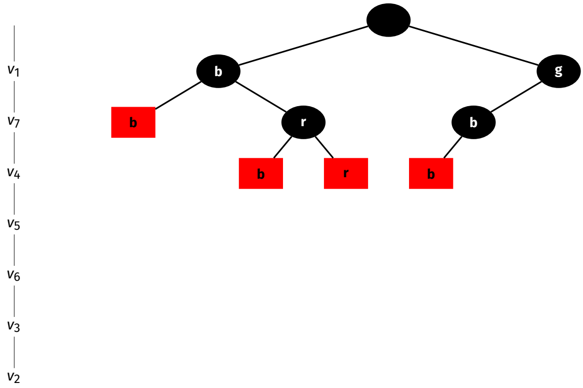
- fixed variable order $v_1, v_7, v_4, v_5, v_6, v_3, v_2$
- alphabetical order of the values



Naive Backtracking: Example

search tree for naive backtracking with

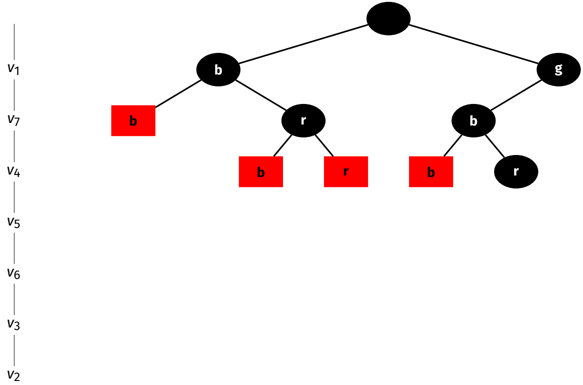
- fixed variable order $v_1, v_7, v_4, v_5, v_6, v_3, v_2$
- alphabetical order of the values



Naive Backtracking: Example

search tree for naive backtracking with

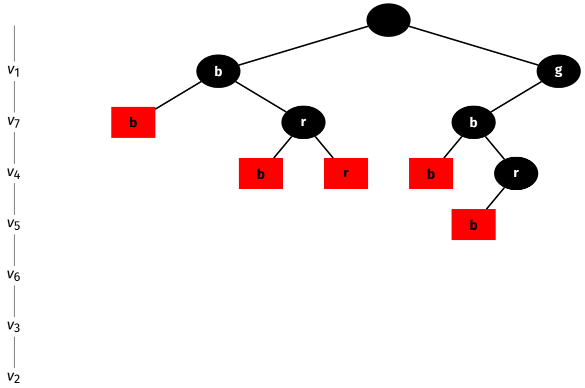
- fixed variable order $v_1, v_7, v_4, v_5, v_6, v_3, v_2$
- alphabetical order of the values



Naive Backtracking: Example

search tree for naive backtracking with

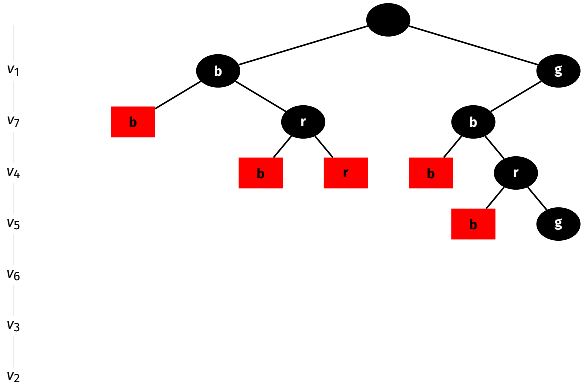
- fixed variable order $v_1, v_7, v_4, v_5, v_6, v_3, v_2$
- alphabetical order of the values



Naive Backtracking: Example

search tree for naive backtracking with

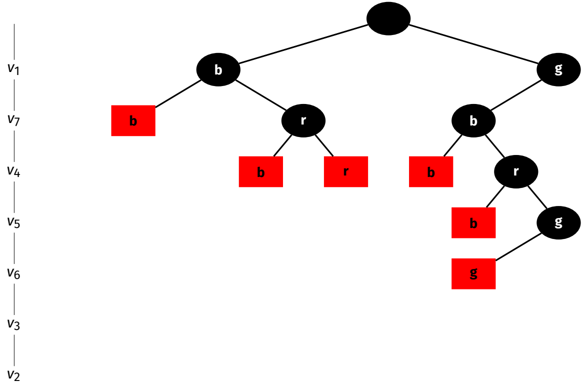
- fixed variable order $v_1, v_7, v_4, v_5, v_6, v_3, v_2$
- alphabetical order of the values



Naive Backtracking: Example

search tree for naive backtracking with

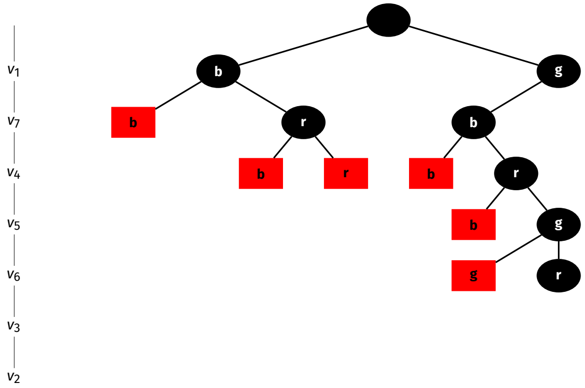
- fixed variable order $v_1, v_7, v_4, v_5, v_6, v_3, v_2$
- alphabetical order of the values



Naive Backtracking: Example

search tree for naive backtracking with

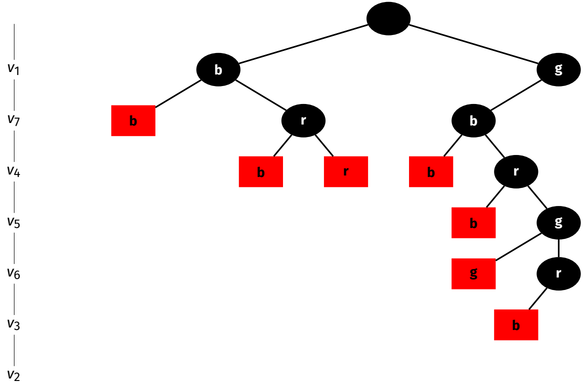
- fixed variable order $v_1, v_7, v_4, v_5, v_6, v_3, v_2$
- alphabetical order of the values



Naive Backtracking: Example

search tree for naive backtracking with

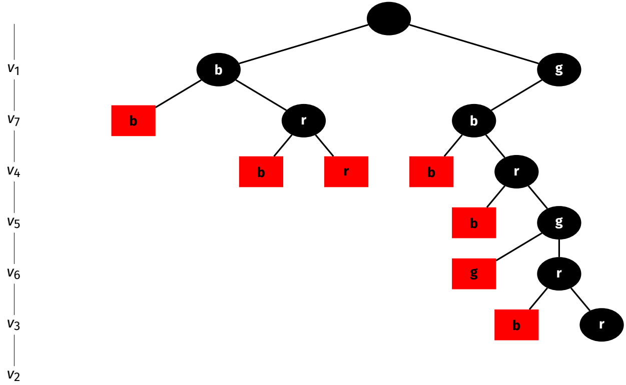
- fixed variable order $v_1, v_7, v_4, v_5, v_6, v_3, v_2$
- alphabetical order of the values



Naive Backtracking: Example

search tree for naive backtracking with

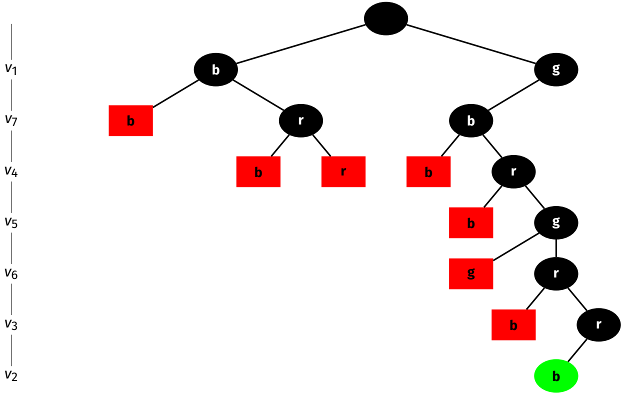
- fixed variable order $v_1, v_7, v_4, v_5, v_6, v_3, v_2$
- alphabetical order of the values



Naive Backtracking: Example

search tree for naive backtracking with

- fixed variable order $v_1, v_7, v_4, v_5, v_6, v_3, v_2$
- alphabetical order of the values



Is This a New Algorithm?

we have already seen this algorithm:
Backtracking corresponds to depth-first search
with the following state space:

- states: partial assignments
- initial state: empty assignment $\emptyset$
- goal states: consistent total assignments
- actions: $\texttt{assign}_{v,d}$ assigns value $d \in dom(v)$ to variable v
- action costs: all 0 (all solutions are of equal quality)
- transitions:
  - for each non-total consistent assignment α,
    choose variable v = SELECT-UNASSIGNED-VARIABLE
  - transition $\alpha \xrightarrow{\texttt{assign}_{v,d}} \alpha \cup \{v \mapsto d\}$ for each $d \in dom(v)$

Why Depth-First Search?

depth-first search is particularly well-suited for CSPs:

- path length bounded (by the number of variables)
- solutions located at the same depth (lowest search layer)
- state space is directed tree, initial state is the root
  $\rightsquigarrow$ no duplicates

hence none of the problematic cases for depth-first search occurs

Naive Backtracking: Discussion

- naive backtracking often has to exhaustively explore similar search paths (i.e., partial assignments that are identical except for a few variables)
- "critical" variables are not recognized and hence considered for assignment (too) late
- decisions that necessarily lead to constraint violations are only recognized when all variables involved in the constraint have been assigned.

$\rightsquigarrow$ more intelligence by focusing on critical decisions and by inference of consequences of previous decisions

Variable and Value Orders

Variable Orders

- Select-Unassigned-Variable method in backtracking search allows to influence order in which variables are considered for assignment
- selected order can strongly influence the search space size and hence the search performance
- general aim: make critical decisions as early as possible

Variable Orders

two common variable ordering criteria:

- minimum remaining values:
  prefer variables that have small domains
  - intuition: few subtrees $\rightsquigarrow$ smaller tree
  - extreme case: only one value $\rightsquigarrow$ forced assignment
- most constraining variable:
  prefer variables contained in many nontrivial constraints
  - intuition: constraints tested early
    $\rightsquigarrow$ inconsistencies recognized early $\rightsquigarrow$ smaller tree

combination: use minimum remaining values criterion,
then most constraining variable criterion to break ties

Value Orders

- ORDER-DOMAIN-VALUES method in backtracking search allows to influence order in which values of the selected variable v are considered
- this is less important because it does not matter in subtrees without a solution
- in subtrees with a solution, ideally a value that leads to a solution should be chosen
- general aim: make most promising assignments first

Value Orders

Definition (conflict)

Let $\mathcal{C} = \langle V, dom, C \rangle$ be a CSP.
For variables $v \neq v'$ and values $d \in dom(v)$, $d' \in dom(v')$,
the assignment $v \mapsto d$ is in conflict with $v' \mapsto d'$ if there is $c \in C$ with
$\mathrm{scope}(c) = (v, v')$ s.t. $(d, d') \notin \mathrm{rel}(c)$.

value ordering criterion for partial assignment α
and selected variable v:

- minimum conflicts: prefer values $d \in dom(v)$
  such that $v \mapsto d$ causes as few conflicts as possible
  with variables that are unassigned in α

Inference

Inference

Inference

Derive additional constraints (here: unary or binary)
that are implied by the given constraints,
i.e., that are satisfied in all solutions.

example: CSP with variables v_1, v_2, v_3 with domain $\{1, 2, 3\}$
and constraints $v_1 < v_2$ and $v_2 < v_3$.

we can infer:

- v_2 cannot be equal to 3 (new unary constraint on v_2)
- $\langle (v1, v2), \{(1, 2), (1, 3), (2, 3)\}$ can be tightened to $\langle (v1, v2), \{(1, 2)\}$ (tighter binary constraint)
- $v_1 < v_3$
  ("new" binary constraint = trivial constraint tightened)

Trade-Off Search vs. Inference

Inference formally

Replace a given CSP C with an equivalent, but tighter CSP.

trade-off:

- the more complex the inference, and
- the more often inference is applied,
- the smaller the resulting state space, but
- the higher the complexity per search node.

When to Apply Inference?

different possibilities to apply inference:

- once as preprocessing before search
- combined with search: before recursive calls during backtracking procedure
  - already assigned variable $v \mapsto d$ corresponds to $dom(v) = \{d\} \rightsquigarrow$ more inferences possible
  - during backtracking, derived constraints have to be retracted because they were based on the given assignment

$\rightsquigarrow$ powerful, but possibly expensive

Backtracking with Inference: Discussion

- INFERENCE method in backtracking search allows to apply different inference methods
- inference methods can recognize unsolvability (given α)
- efficient implementations of inference are often incremental:
  the last assigned variable/value pair $v \mapsto d$ is taken
  into account to speed up the inference computation

Node Consistency

Node Consistency

We start with a simple inference method:

Node Consistency

Remove all values from the domain of all variable v that are in conflict with a unary constraint on v.

Node Consistency: Discussion

properties of node consistency:

- correct inference method (retains equivalence)
- affects domains (= unary constraints), but not binary constraints
- cheap, but often still useful inference method

$\rightsquigarrow$ minimal inference method that should (almost) always be used

Arc Consistency

Arc Consistency: Definition

Definition (Arc Consistent)

Let $\mathcal{C} = \langle V, dom, C\rangle$ be a CSP.

(a) A variable $v \in V$ is arc consistent
with respect to another variable $v' \in V$,
if for every value $d \in dom(v)$
there exists a value $d' \in dom(v')$ with $\langle d, d'\rangle \in c$ with
$\mathrm{scope}(c) = (v, v')$

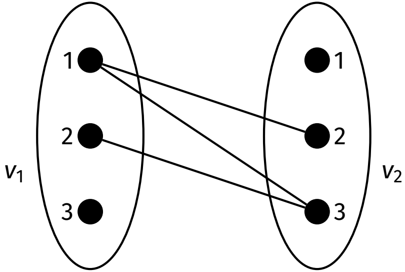
(b) The CSP $\mathcal{C}$ is arc consistent,
if every variable $v \in V$ is arc consistent
with respect to every other variable $v' \in V$.

remarks:

- definition for variable pair is not symmetrical
- v always arc consistent with respect to v'
  if the constraint between v and v' is trivial

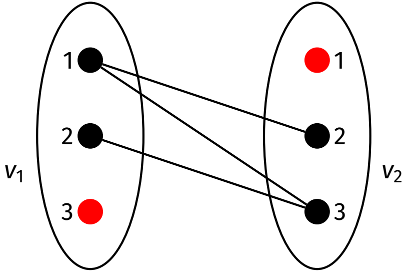
Arc Consistency: Example

Consider a CSP with variables v_1 and v_2,
domains $dom(v_1) = dom(v_2) = \{1, 2, 3\}$
and the constraint expressed by $v_1 < v_2$.



Arc Consistency: Example

Consider a CSP with variables v_1 and v_2,
domains $dom(v_1) = dom(v_2) = \{1, 2, 3\}$
and the constraint expressed by $v_1 < v_2$.



Arc consistency of v_1 with respect to v_2
and of v_2 with respect to v_1 are violated.

Enforcing Arc Consistency

- enforcing arc consistency, i.e., removing values from $dom(v)$ that violate the arc consistency of v with respect to v', is a correct inference method
- more powerful than node consistency:
  - $\rightsquigarrow$ node consistency is a special case: enforcing arc consistency of all variables with respect to the just assigned variable corresponds to node consistency.

Processing Variable Pairs: REVISE

function REVISE($\langle V, dom, C\rangle, v, v'$):

    revised = **false**

    let $c = \langle (v, v'), \mathrm{rel}\rangle \in C$

    for each $d \in dom(v)$:

        if there is no $d' \in dom(v')$ s.t. $(d, d') \in \mathrm{rel}(c)$:

            remove d from $dom(v)$

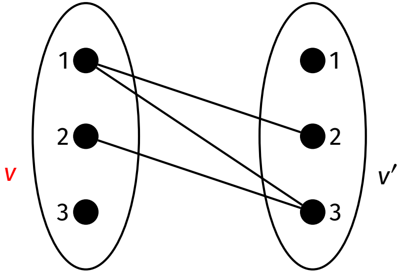
            revised = **true**

    return *revised*

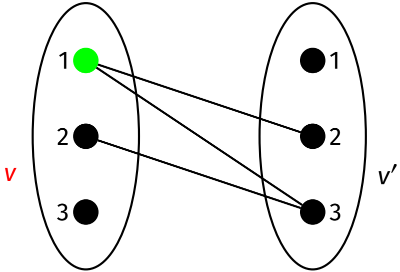
effect: v arc consistent with respect to v'.
All violating values in $dom(v)$ are removed.

time complexity: $O(k^2)$, where k is maximal domain size

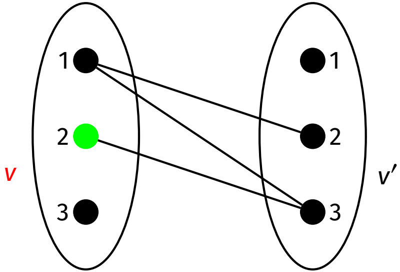
Example: revise



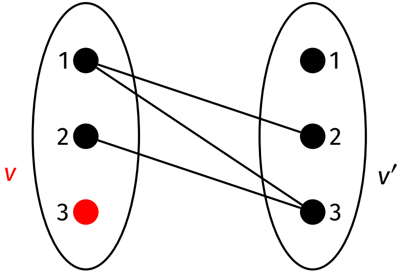
Example: revise



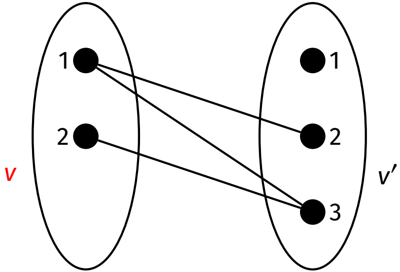
Example: revise



Example: revise



Example: revise



Enforcing Arc Consistency: AC-3

idea:

- transform $\mathcal{C}$ into equivalent arc consistent CSP
- store potentially inconsistent variable pairs in a queue

function AC-3$(\mathcal{C})$:

$\langle V, dom, C\rangle := \mathcal{C}$

queue $:= \emptyset$

for each nontrivial constraint c with scope$(c) = (u, v)$:

  insert $\langle u, v\rangle$ into *queue*

  insert $\langle v, u\rangle$ into *queue*

while *queue* $\neq \emptyset$:

  remove an arbitrary element $\langle u, v\rangle$ from *queue*

  if REVISE$(\mathcal{C}, u, v)$:

    for each $w \in V \setminus \{u, v\}$ where c is nontrivial:

      insert $\langle w, u\rangle$ into *queue*

Path Consistency

Path Consistency

idea of arc consistency:

- for every assignment to a variable u
  there must be a suitable assignment to every other variable v
- If not: remove values of u for which
  no suitable "partner" assignment to v exists

$\rightsquigarrow$ tighter unary constraint on u

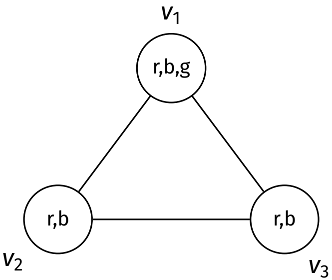
this idea can be extended to three variables (path consistency):

- for every joint assignment to variables u, v
  there must be a suitable assignment to every third variable w
- if not: remove pairs of values of u and v for which
  no suitable "partner" assignment to w exists.

$\rightsquigarrow$ tighter binary constraint on u and v

Path Consistency: Example

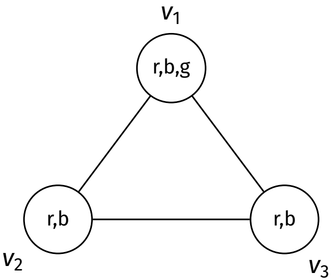
arc consistent, but not path consistent



$$c_{12} = \langle (v_1, v_2), \{(r, b), (b, r), (g, r), (g, b)\} \rangle$$
$$c_{13} = \langle (v_1, v_3), \{(r, b), (b, r), (g, r), (g, b)\} \rangle$$
$$c_{23} = \langle (v_2, v_3), \{(r, b), (b, r)\} \rangle$$

Path Consistency: Example

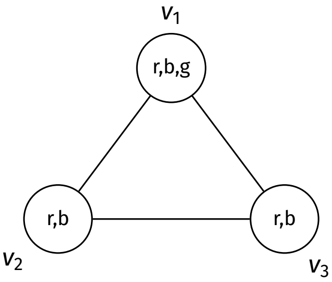
arc consistent, but not path consistent



$$c_{12} = \langle (v_1, v_2), \{(r, b), (b, r), (g, r), (g, b)\}$$
$$c_{13} = \langle (v_1, v_3), \{(r, b), (b, r), (g, r), (g, b)\}$$
$$c_{23} = \langle (v_2, v_3), \{(r, b), (b, r)\}$$

Path Consistency: Example

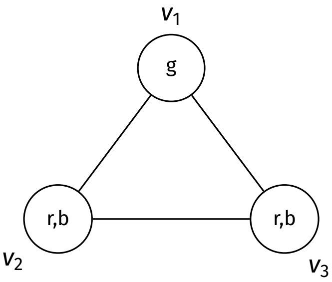
not arc consistent, but path consistent



$$c_{12} = \langle (v_1, v_2), \{(g, r), (g, b)\} \rangle$$
$$c_{13} = \langle (v_1, v_3), \{(g, r), (g, b)\} \rangle$$
$$c_{23} = \langle (v_2, v_3), \{(r, b), (b, r)\} \rangle$$

Path Consistency: Example

arc consistent and path consistent



$$c_{12} = \langle (v_1, v_2), \{(g, r), (g, b)\} \rangle$$
$$c_{13} = \langle (v_1, v_3), \{(g, r), (g, b)\} \rangle$$
$$c_{23} = \langle (v_2, v_3), \{(r, b), (b, r)\} \rangle$$