Fundamentals of Artificial Intelligence
Exercise 3: Constraint Satisfaction Problems

Sebastian Mair

Technical University of Munich

November 17th, 2023

Recap: Constraint Satisfaction Problems (CSP)

CSP tuple (X, D, C)

- $X = \{X_1, \ldots, X_n\}$ is a set of variables
- $D = \{D_1, \ldots, D_n\}$ is a set of domains, where each domain D_i is the set of allowable values $\{v_1, \ldots, v_k\}$ for variable X_i
- $C = \{C_1, \ldots, C_m\}$ is a set of constraints (can be **unary**, **binary** or **higher-order (n-ary)**)

Goal: Assign a value to each variable such that all constraints are satisfied

Recap: Backtracking Search

function Recursive-Backtracking (*assignment*, *csp*) **returns** sol./failure

```
if assignment is complete then return assignment
var ← Select-Unassigned-Variable(csp)
for each value in Order-Domain-Values(var, assignment, csp) do
    if value is consistent with assignment given Constraints[csp] then
        add {var = value} to assignment
        inferences ← Inference(csp,var,value)
        if inferences ≠ failure then
            add inferences to assignment
            result ← Recursive-Backtracking(assignment,csp)
            if result ≠ failure then return result
            remove inferences from assignment
        remove {var = value} from assignment
return failure
```

Important heuristics:

- Variable Selection
- Value Selection
- Inference

Recap: Variable Selection and Value Selection Heuristics

Variable Selection

- **Minium Remaining Values** (MRV): choose variable with the fewest possible values first
- **Degree heuristics**: choose variable that is involved in the largest number of constraints on other unassigned variables (highest degree)

Value Selection

- **Least Constraining Value**: choose value that rules out the fewest choices for neighboring values in the constraint graph

Recap: Inference in CSP

Inference techniques

- **Forward checking** (after each assignment): inconsistent values of neighboring variables are removed.
- **Arc consistency algorithm** (after each assignment or as pre-processing): inconsistent values of all variables are removed.

Arc consistency of a variable

X_i is arc-consistent with X_j, if for every value in the domain D_i there exists a value in D_j satisfying the binary constraint of the arc (X_i, X_j).

Example: Constraint: $Y = X^2$, Domains $D_X = \{0, 1, 2, 3\}$ and $D_Y = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
$\rightarrow$ Question: (X, Y) and/or (Y, X) arc-consistent?

Tweedback code: **znnk** (twbk.de/znnk)

Arc Consistency Algorithm

function AC-3 (*csp*, *queue*) **returns** *failure* or the reduced *csp* otherwise

inputs: *csp*: a binary CSP, *queue*: a queue of arcs (X_i, X_j)
while *queue* is not empty **do**
    $(X_i, X_j) \leftarrow$ `Remove-First`(*queue*)
    if `Remove-Inconsistent-Values`(X_i, X_j) **then**
        if size of `Domain`$(X_i) = 0$ **then return** *failure*
        for each X_k **in** `Neighbors`$[X_i] \setminus \{X_j\}$ **do**
            add (X_k, X_i) to *queue*
return *csp*

function Remove-Inconsistent-Values (X_i, X_j) **returns** true iff succeeds

removed $\leftarrow$ *false*
for each x **in** `Domain`$[X_i]$ **do**
    if no value y in `Domain`$[X_j]$ allows (x,y) to satisfy the constraint of (X_i, X_j)
        then delete x from `Domain`$[X_i]$; *removed* $\leftarrow$ *true*
return *removed*

Problem 3.1: Turning n-ary constraints into binary constraints

Suppose that we have $CSP = (X, D, E^1)$ with

$$\begin{aligned} X &= \{A, B, C\}, \\ D &= \{\mathrm{dom}(A), \mathrm{dom}(B), \mathrm{dom}(C)\}, \\ E &= \{\langle (A, B, C),\ A + B = C\rangle\}, \end{aligned}$$

where each domain can be $\{0, 1, \ldots, 9\}$ for example.

[1] the symbol E is taken from German word *Einschränkung*.

Problem 3.1.1

Draw the constraint hypergraph for the CSP. Based on the number of variables involved, what is the type of the constraint?

$$\begin{aligned} X &= \{A, B, C\}, \\ D &= \{\mathrm{dom}(A), \mathrm{dom}(B), \mathrm{dom}(C)\}, \\ E &= \{\langle (A, B, C),\ A + B = C \rangle\}, \end{aligned}$$

Problem 3.1.2

We can eliminate the higher-order constraint in E by replacing the constraint node $\square$ with a new variable node Z. What is the domain for variable Z? What is the new constraint set E' after introducing the new variable Z?

$$
\begin{aligned}
X &= \{A, B, C\}, \\
D &= \{\mathrm{dom}(A), \mathrm{dom}(B), \mathrm{dom}(C)\}, \\
E &= \{\langle (A, B, C),\ A + B = C \rangle\},
\end{aligned}
$$

$\mathrm{dom}(Z) = \{ z_1, z_2, z_3 \mid z_1 = \mathrm{dom}(A) \wedge z_2 = \mathrm{dom}(B) \wedge z_3 = \mathrm{dom}(C) \}$

$E' = \{ \langle (z_1, A), z_1 = A \rangle \ \langle (z_2, B), z_2 = B \rangle \ \langle (z_3, C), z_3 = C \rangle \} \cup E$

Problem 3.1.3

Modify CSP' such that it only contains binary constraints and formally express the new $CSP'' = (X'', D'', E'')$.

$$
\begin{aligned}
X &= \{A, B, C\},\\
D &= \{\mathrm{dom}(A), \mathrm{dom}(B), \mathrm{dom}(C)\},\\
E &= \{\langle (A, B, C),\ A + B = C\rangle\},
\end{aligned}
$$

$X'' = \{A, B, C, Z\}$

$D'' = D \cup (\mathrm{dom}(Z))$

$\mathrm{dom}(Z) = \{z_1, z_2, z_3 \mid z_1 = \mathrm{dom}(A) \wedge z_2 = \mathrm{dom}(B) \wedge z_3 = \mathrm{dom}(C$

$\wedge\ z_1 + z_2 = z_3\}$

$E'' = \{\langle z_1 = A\rangle, \langle z_2 = B\rangle\ \langle z_3 = C\rangle\}$

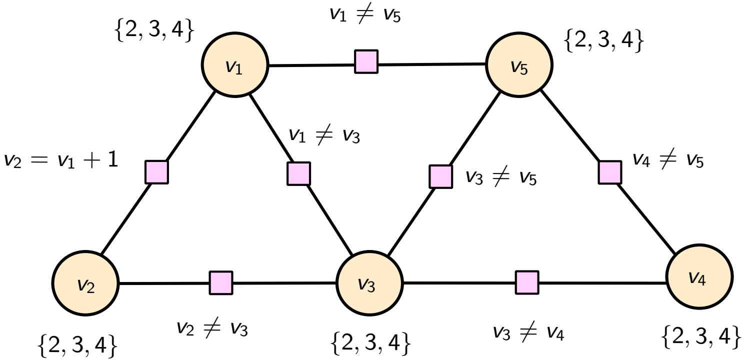
binary

Problem 3.1.4

Taking inspiration from previous solutions, how can you generally turn a n-ary constraint into binary constraints?

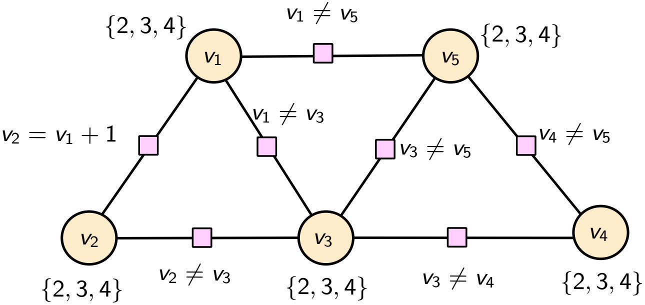
Problem 3.2

Consider the constraint satisfaction problem in the figure below.



$\rightarrow$ only binary constraints.

Problem 3.2.1: Sort variables by domain size and by degree



domain size

3 V_1 V_2 V_3 V_4 V_5

degree size

3 V_1 V_5

2 V_2 V_4

4 V_3

Problem 3.2.2: Perform Backtracking Search by Hand

Variable selection:
Variable to expand next: apply minimum-remaining-values (MRV) heuristic; if there is a tie, use degree heuristics; if there is a tie again, choose the variable with the lower index.

Value selection:
Value to assign next: use least-constraining-value heuristics; if there is a tie, choose number 3; if this is not possible choose the lowest value.

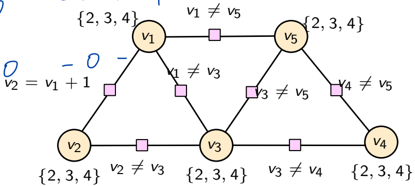
Inference:
After each assignment: perform forward checking as inference. Backtrack if you find an inconsistency.

Problem 3.2.2: Perform Backtracking Search by Hand

step	assign	current domains v_1	v_2	v_3	v_4	v_5	degree v_1	v_2	v_3	v_4	v_5	*backtrack to*
0	/	234	234	234	234	234	3	2	4	2	3	
1	$\boxed{V_3}$ degree =3	24	24	/	24	24	2	1	–	1	2	
2	$\boxed{V_1}$ low index =2	/	∅	/				.	.	–		2
2	$\boxed{V_1}$=4	/	∅	/								
1	$\boxed{V_3}$=2	34	34	/	34	34	2	1	–	1	2	
2	V_1=3	/	~~4~~	/	34	4	–	0	–	1	1	
3	V_5=4 tie	/	4	/	3	/	–	0	–	0	–	

Variable: MRV → Degree → lower index
Value: least constr. value → 3 → lowest value
Inference: forward checking
Tweedback code: **znnk** (twbk.de/znnk)

4 $V_2 = 4$
5 $V_4 = 3$



Problem 3.2.3: Perform Backtracking Search by Hand

Variable selection:
Variable to expand next: apply minimum-remaining-values (MRV) heuristic; if there is a tie, use degree heuristics; if there is a tie again, choose the variable with the lower index.

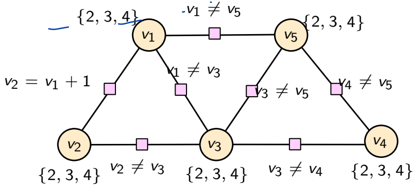
Value selection:
Value to assign next: use least-constraining-value heuristics; if there is a tie, choose number 3; if this is not possible choose the lowest value.

Inference:
After each assignment: perform arc consistency algorithm. Backtrack if you find an inconsistency.

Problem 3.2.3: Perform Backtracking Search by Hand

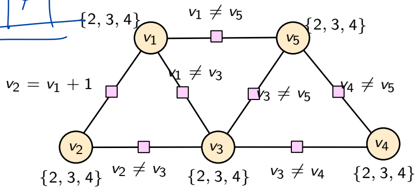
		current domains					degree					backtrack to
step	assign	v_1	v_2	v_3	v_4	v_5	v_1	v_2	v_3	v_4	v_5	
0	/	234	234	234	234	234	3	2	4	2	3	
1	$v_3=3$	24	24	/	24	24	2	1	-	1	2	
1	$v_3=2$	3	4	/	3	4	2	1	-	1	2	1
2	$v_1=3$	/	4	/	3	4	-	0	-	1	1	
3	$v_4=3$	/	4	/	/	4	-	0	-	-	0	
4	$v_2=4$	/	/	/	/	4	-	-	-	-	0	
5	$v_5=4$	/	/	/	/	/	-	-				

Variable: MRV → Degree → lower index
Value: least constr. value → 3 → lowest value
Inference: arc consistency algorithm



Problem 3.2.3: Perform Backtracking Search by Hand (AC-3)

Arc	Queue	Domain Change
	~~(V_1, V_3)~~ ~~(V_2, V_3)~~ ~~(V_4, V_3)~~ ~~(V_5, V_3)~~	
(V_1, V_3)	~~(V_2, V_1)~~ (V_5, V_1) (V_1, V_2) (V_5, V_4)	V_1 : 2 4
(V_2, V_1)	(V_1, V_5) (V_4, V_5)	V_2 : 2 4
(V_4, V_3)		V_4 : 2 4
(V_5, V_3)		V_5 : 2 4
(V_2, V_1)		V_2 : $\boxed{\emptyset}$

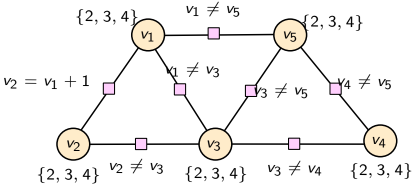


Tweedback code: **znnk** (twbk.de/znnk)

Problem 3.2.3: Perform Backtracking Search by Hand (AC-3)

Arc	Queue	Domain Change

Tweedback code: **znnk** (twbk.de/znnk)

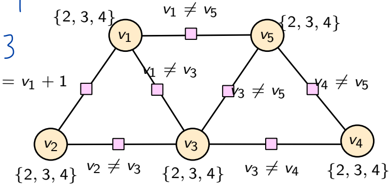


Problem 3.2.3: Perform Backtracking Search by Hand (AC-3)

Arc	Queue	Domain Change
	~~(V1, V3)~~ ~~(V2, V3)~~ ~~(V4, V3)~~ ~~(V5, V3)~~	
(V1, V3)	~~(V2, V1)~~ ~~(V3, V1)~~ ~~(V1, V2)~~ ~~(V5, V4)~~	~~V1 : 3 4~~
(V2, V1)	~~(V1, V5)~~ ~~(V4, V5)~~ ~~(V3, V2)~~ ~~(V3, V1)~~	~~V2 : 3 4~~
(V4, V3)	~~(V5, V1)~~ ~~(V3, V5)~~ ~~(V3, V4)~~ ~~(V4, V5)~~	~~V4 = 3 4~~
(V5, V3)		~~V5 = 3 4~~
(V2, V1)		V2 = 4
(V5, V1)		V1 = 3
(V1, V2)		V5 = 4
(V5, V4)		V4 = 3
(V1, V5)		

(V4 V5)
(V3 V2)
(V3 V1)

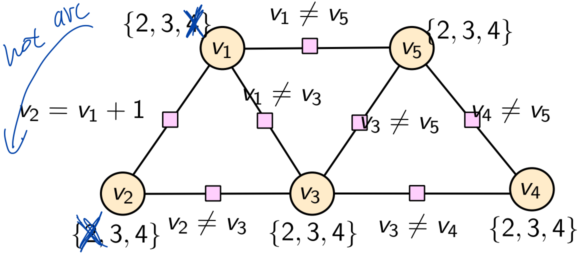
(V5 V1)
(V3, V5)
(V3, V4)



Tweedback code: **znnk** (twbk.de/znnk)

Problem 3.2.4

Consider the constraint satisfaction problem at its initial state. Is the CSP arc consistent? Is this a convenient initial condition if we plan to apply backtracking search? Describe the domains after the arc consistency algorithm has been applied to the CSP as a preprocessing step.



Tweedback code: **znnk** (twbk.de/znnk)

Problem 3.2.4

New, arc-consistent CSP:

Problem 3.2.5: Backtracking Search after Preprocessing

Variable selection:
Variable to expand next: apply minimum-remaining-values (MRV) heuristic; if there is a tie, use degree heuristics; if there is a tie again, choose on randomly.

Value selection:
Value to assign next: use least-constraining-value heuristics; if there is a tie, choose choose one randomly.

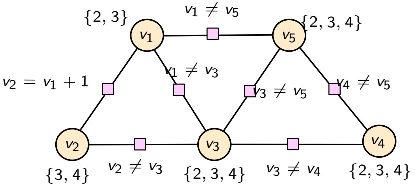
Inference:
After each assignment: perform arc consistency algorithm. Backtrack if you find an inconsistency.

Assume the data structure of the queue is a set, i.e., if we add an element to the queue which is already in the queue, the element will not be added a second time (each element is unique).

Problem 3.2.5: Backtracking Search after Preprocessing

		current domains					degree					*backtrack to*
step	assign	v_1	v_2	v_3	v_4	v_5	v_1	v_2	v_3	v_4	v_5	
0	∅	23	34	234	234	234	3	2	4	2	3	
1	$V_1=2$											

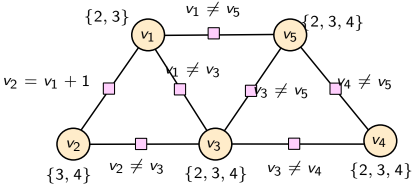
Variable: MRV → Degree → randomly
Value: least constr. value → randomly
Inference: arc consistency algorithm
Tweedback code: **znnk** (twbk.de/znnk)



Problem 3.2.5: Backtracking Search after Preprocessing (AC-3)

$V_1 = 2$

Arc	Queue	Domain Change
	~~(V_2, V_1)~~ ~~(V_3, V_1)~~ ~~(V_5, V_1)~~	$V_2 = 3$
(V_2, V_1)	~~(V_3, V_2)~~ ~~(V_2, V_3)~~ ~~(V_5, V_3)~~ ~~(V_4, V_3)~~	~~$V_3 = 3\ 4$~~
(V_3, V_1)	(V_3, V_5) $(V_4\ V_5)$ (V_3, V_1) (V_1, V_2)	$V_5 = 3\ \ 4$
$(V_5\ V_1)$		
(V_3, V_2)	(V_4, V_3)	$V_3 = 4$
(V_2, V_3)		
(V_5, V_3)		



Tweedback code: **znnk** (twbk.de/znnk)

Problem 3.2.6

For each of the previous performances of backtracking search with a different inference compare the number of iterations and the number of times you needed to backtrack.

Forward checking (3.2.2): ____________________

Arc Consistency Algorithm (3.2.3): ____________________

Arc Consistency Algorithm after preprocessing (3.2.5): ____________________