

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)

- $X = \{X_1, \dots, X_n\}$ is a set of variables
- $D = \{D_1, \dots, D_n\}$ is a set of domains, where each domain D_i is the set of allowable values $\{v_1, \dots, v_k\}$ for variable X_i
- $C = \{C_1, \dots, C_m\}$ 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  $\leftarrow$  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  $\{ \text{var} = \text{value} \}$  to assignment
      inferences  $\leftarrow$  Inference(csp, var, value)
      if inferences  $\neq$  failure then
        add inferences to assignment
        result  $\leftarrow$  Recursive-Backtracking(assignment, csp)
        if result  $\neq$  failure then return result
        remove inferences from assignment
      remove  $\{ \text{var} = \text{value} \}$  from assignment
  return failure

```

Important heuristics:

- Variable Selection
- Value Selection
- Inference

Recap: Variable Selection and Value Selection Heuristics

Variable Selection

- **Minimum 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.

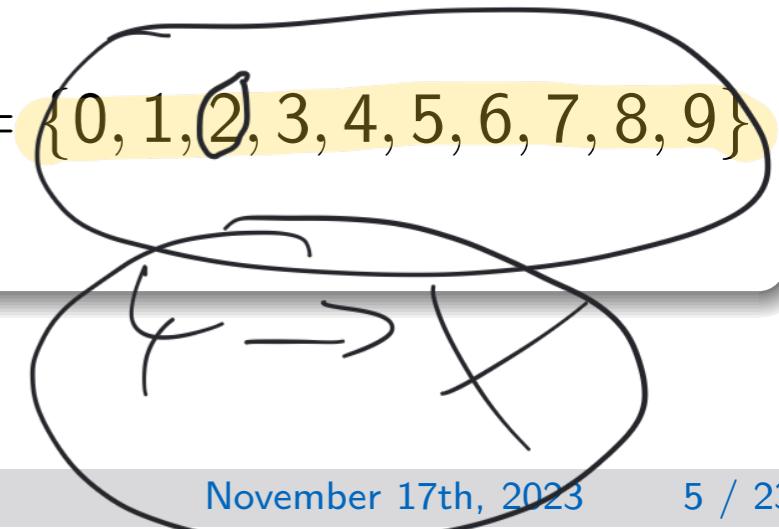
Arc consistency of a variable

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

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

X, f



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**

$(X_i, X_j) \leftarrow \text{Remove-First}(queue)$

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] \{ 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

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

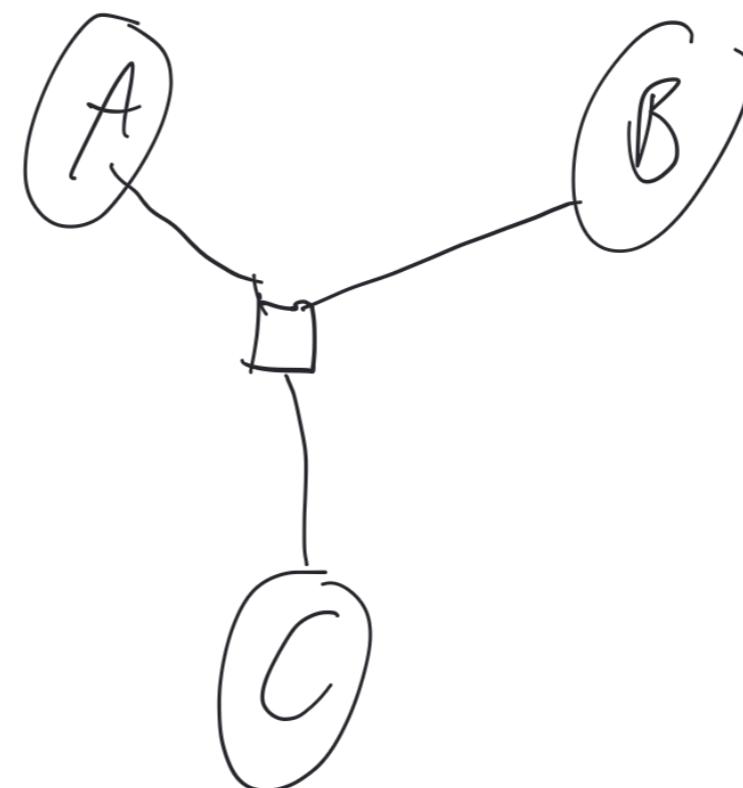
where each domain can be $\{0, 1, \dots, 9\}$ for example.

¹the 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?

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



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 ? $CSP/$

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

~~A, B, C~~

$$\text{dom}(Z) = \left\{ (z_1, z_2, z_3) \mid z_1 \in \text{dom}(A) \wedge z_2 \in \text{dom}(B) \wedge z_3 \in \text{dom}(C) \right\}$$

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

Problem 3.1.3

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

$$\begin{aligned} X &= \{A, B, C\}, \\ D &= \{\text{dom}(A), \text{dom}(B), \text{dom}(C)\}, \\ E &= \underbrace{\{(A, B, C), A + B = C\}}_{\mathcal{Z}}, \end{aligned}$$

 \mathcal{Z}

$$\begin{aligned} \text{dom}(\mathcal{Z}) &= \left\{ (z_1, z_2, z_3) \mid \right. \\ &\quad z_1 \in \text{dom}(A) \\ &\quad \wedge z_2 \in \text{dom}(B) \\ &\quad \wedge z_3 \in \text{dom}(C) \\ &\quad \left. \wedge z_1 + z_2 = z_3 \right\} \end{aligned}$$

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

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

$$E'' = \left\{ z_1 = A, z_2 = B, z_3 = C \right\}$$

$$\underbrace{\langle (A, \mathcal{Z}), t = z_0 \rangle}_{f_{st}(\mathcal{Z})},$$

$$f_{st}(\mathcal{Z})$$

Problem 3.1.4

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



Problem 3.1.4

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

Suppose that we have a constraint of order $n \rightarrow$ can be represented by a relation $R(x_1, x_2, \dots, x_n)$ of order n .

Problem 3.1.4

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

Suppose that we have a constraint of order $n \rightarrow$ can be represented by a relation $R(x_1, x_2, \dots, x_n)$ of order n .

- ① We replace the n -ary constraint by a new variable Z . The domain of Z is a n -tuple and must be restricted such that it satisfies the relation R .

Problem 3.1.4

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

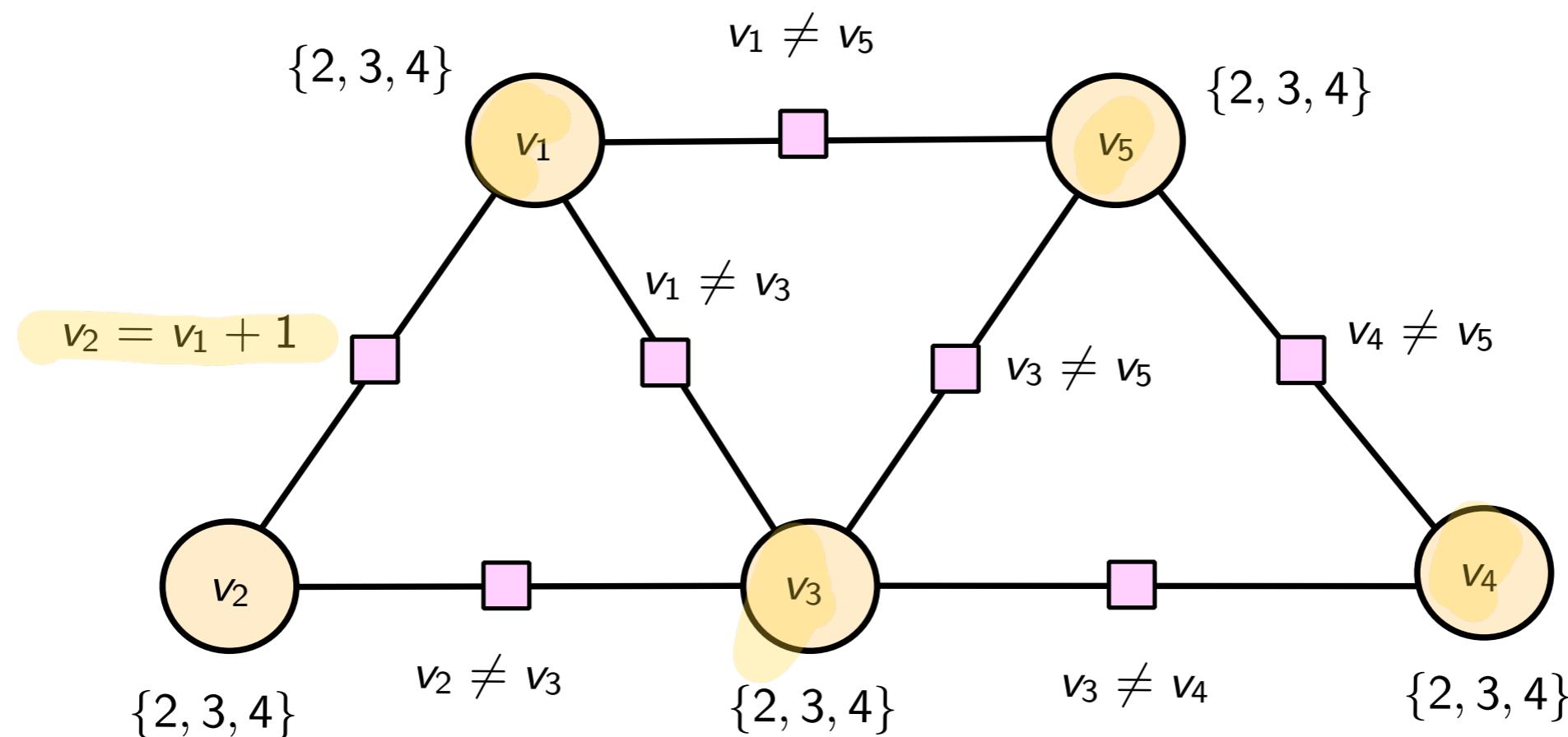
Suppose that we have a constraint of order $n \rightarrow$ can be represented by a relation $R(x_1, x_2, \dots, x_n)$ of order n .

- ① We replace the n -ary constraint by a new variable Z . The domain of Z is a n -tuple and must be restricted such that it satisfies the relation R .
- ② Then, we introduce new binary constraints to match the values of variable Z with the values of the neighboring variables x_1, x_2, \dots, x_n (i.e., $\text{fst}(Z) = x_1$, $\text{snd}(Z) = x_2$, and so on).

$$\begin{array}{c} z_1 = x_1 \\ \vdots \end{array}$$

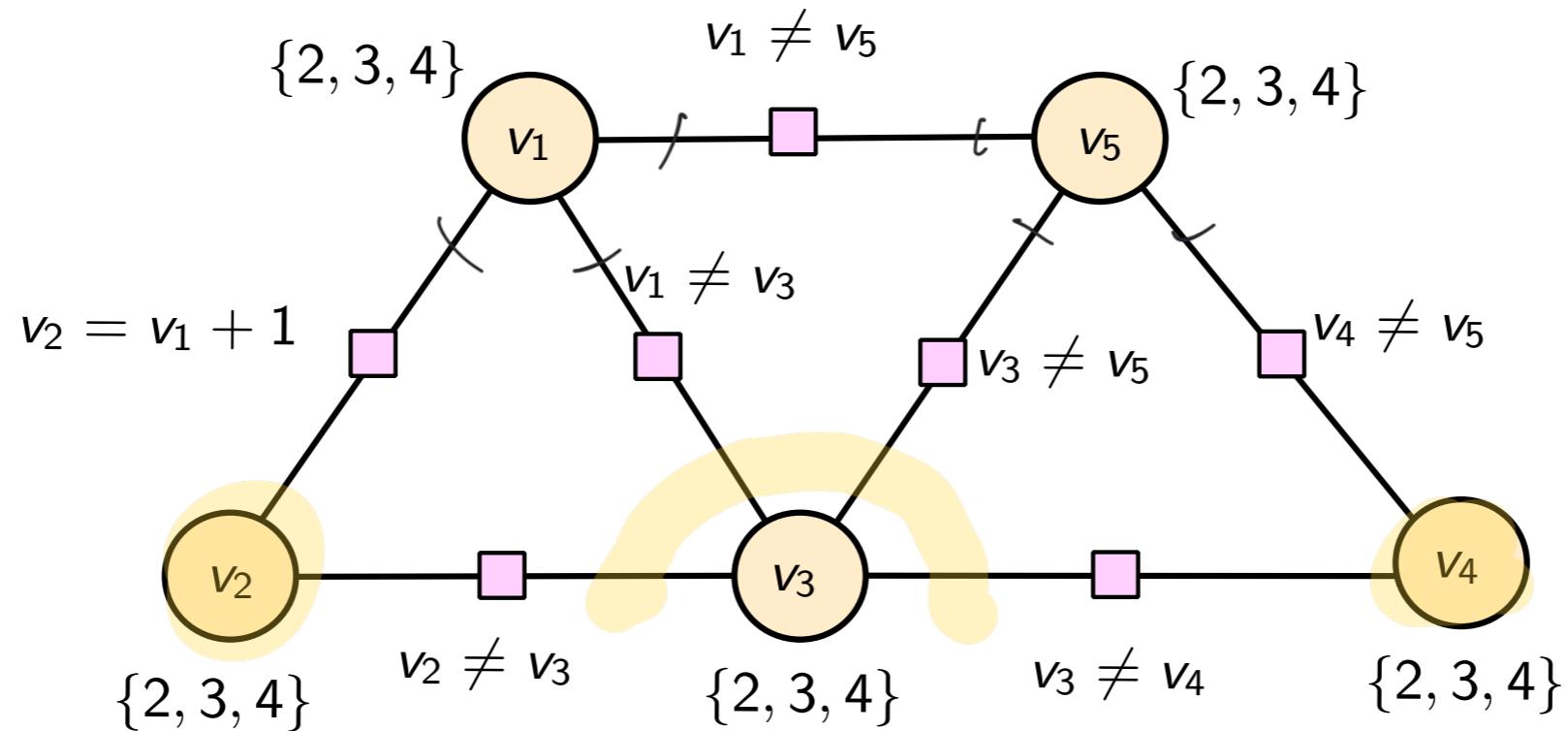
Problem 3.2

Consider the constraint satisfaction problem in the figure below.



→ only binary constraints.

Problem 3.2.1: Sort variables by domain size and by degree



Domain size: 3

2 : v_2, v_4

Degree :

3 : v_1, v_5

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

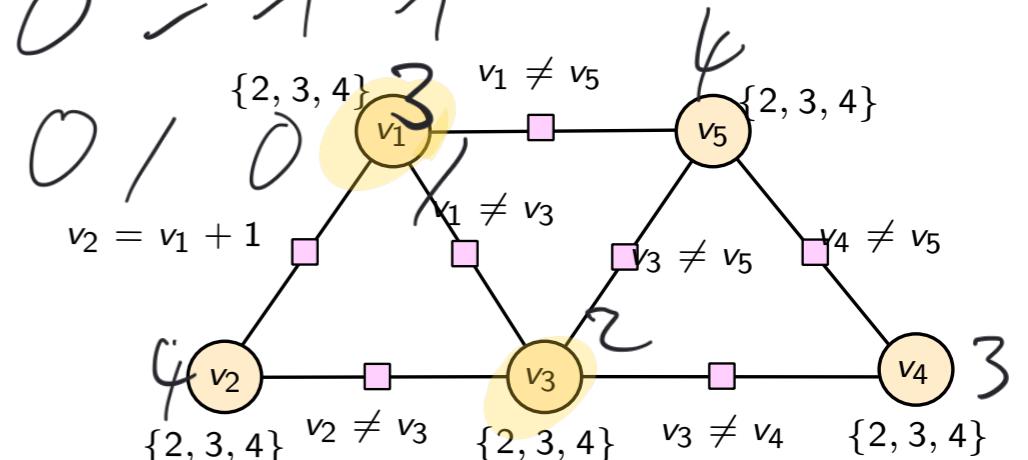
step	assign	current domains					degree					backtrack to
		v_1	v_2	v_3	v_4	v_5	v_1	v_2	v_3	v_4	v_5	
0	/	234	234	234	234	234	3	2	4	2	3	
1	$v_3 = 3$	24	24	24	24	24	2	1	1	1	2	*
2	$v_1 = 2$	/	(0)	/	24	4	-	-	-	-	-	2
2	$v_1 = 4$	/	(0)	/	24	2	-	-	-	-	-	1
1	$v_3 = 2$	34	/	34	34	2	2	1	1	2		
2	$v_1 = 3$	/	4	/	34	4	-	0	-	1	1	
3	$v_5 = 4$	/	4	/	3	/	/	0	0	1	4	

Variable: MRV \rightarrow Degree \rightarrow lower index

Value: least constr. value \rightarrow 3 \rightarrow 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.

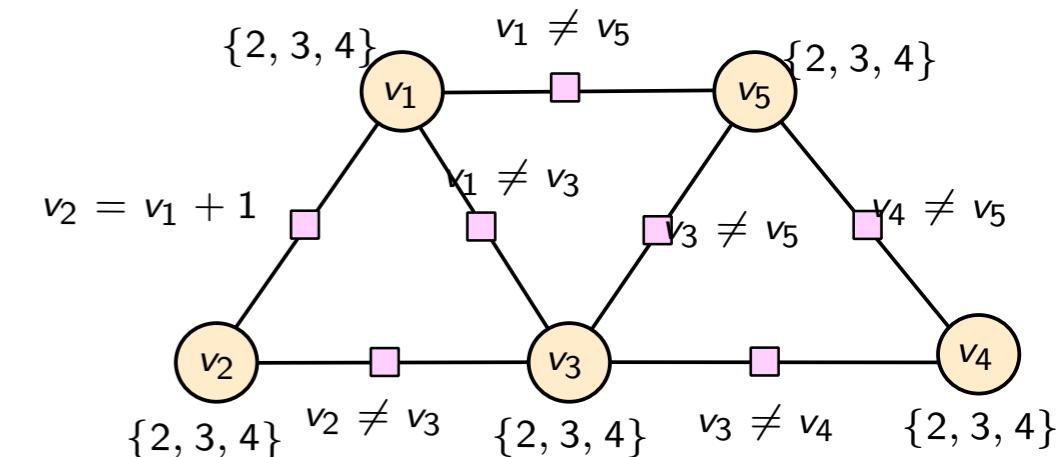
Problem 3.2.3: Perform Backtracking Search by Hand

step	assign	current domains					degree					backtrack to
		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	$v_3 = 3$											

Variable: MRV \rightarrow Degree \rightarrow lower index

Value: least constr. value \rightarrow 3 \rightarrow lowest value

Inference: arc consistency algorithm

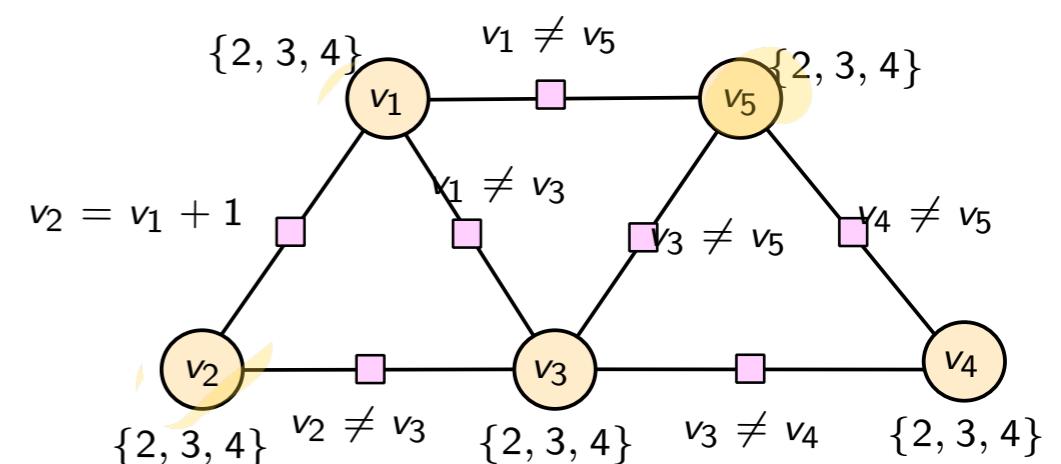


$v_3 = 3$

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

Arc	Queue	Domain Change
(v_1, v_3)	$(v_1, v_3), (v_2, v_3), (v_4, v_3), (v_5, v_3)$	
(v_2, v_3)	$(v_2, v_1), (v_5, v_1), (v_1, v_2)$	$v_1 : 2, 4$ $v_2 : 2, 4$
(v_4, v_3)		$v_4 : 2, 4$

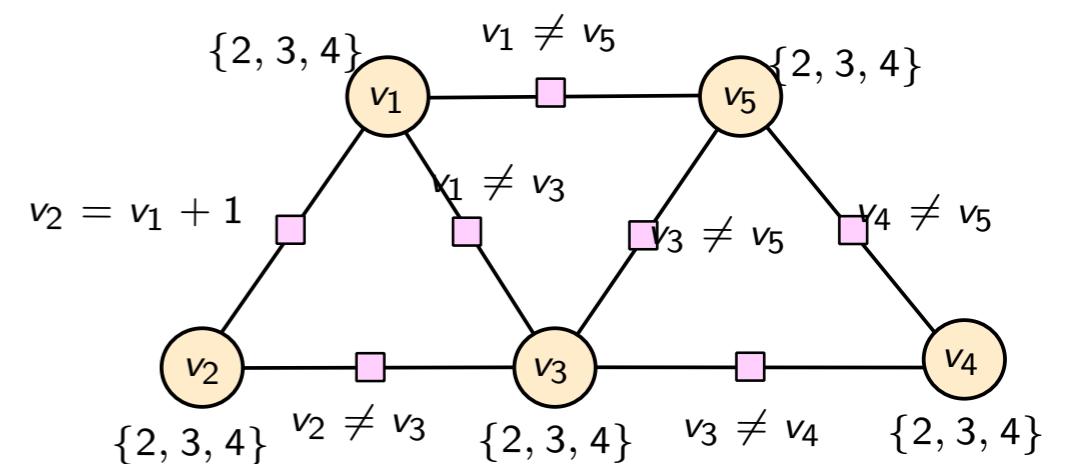
Tweedback code: **znnk** (twbk.de/znnk)



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

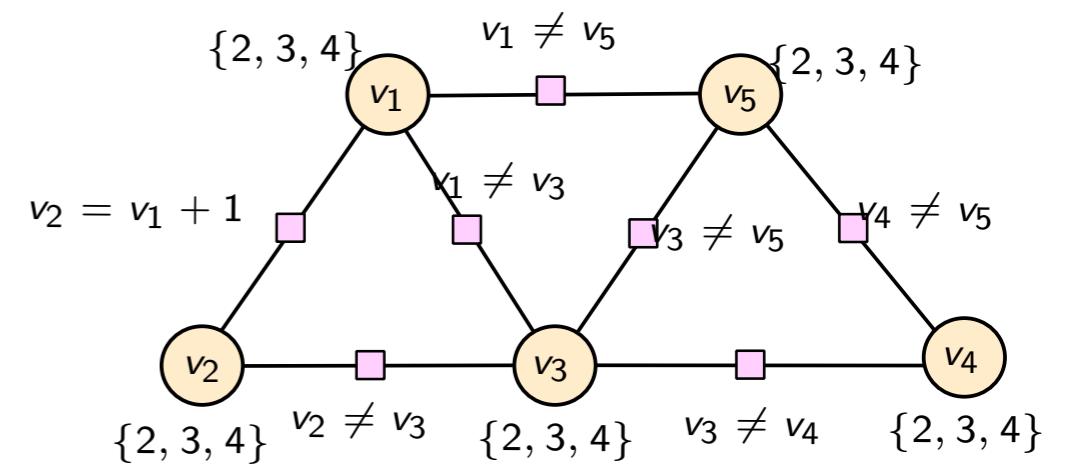
Arc	Queue	Domain Change
-----	-------	---------------

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



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

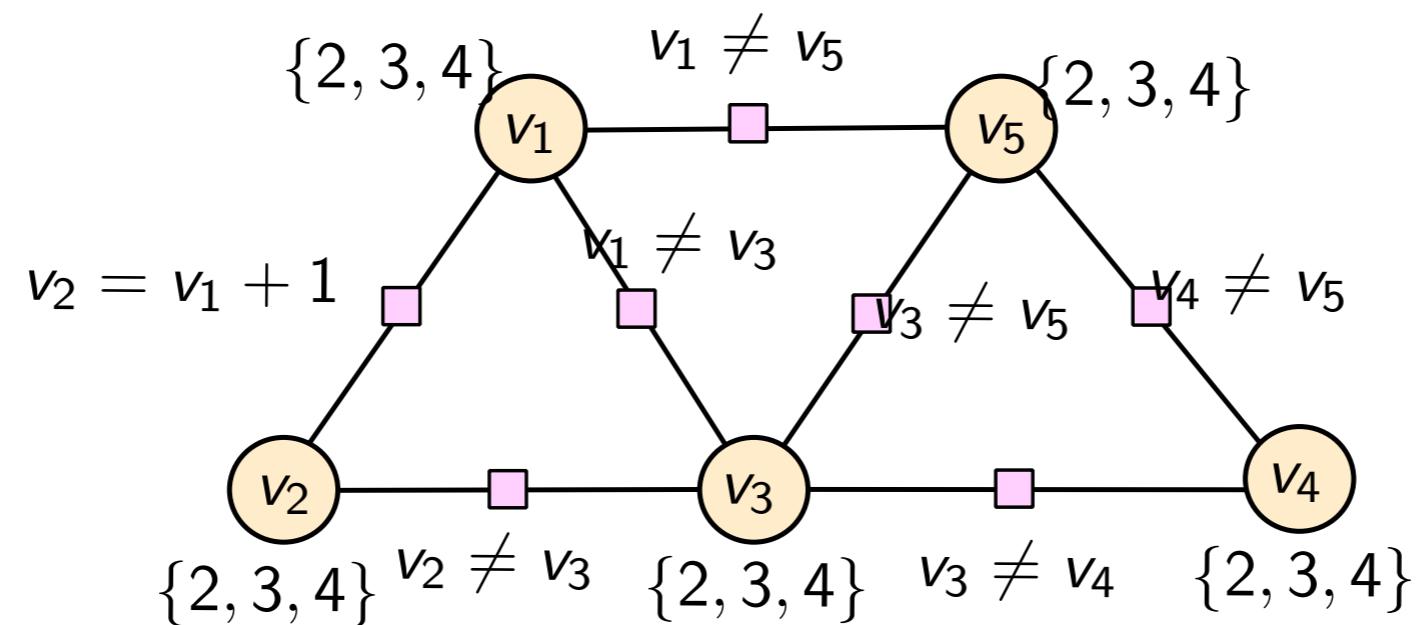
The diagram illustrates a process flow or algorithmic step. It features a horizontal line with three main sections: "Arc" on the left, "Queue" in the center, and "Domain Change" on the right. The "Queue" section is currently empty. The "Domain Change" section shows a graph node labeled v_2 with an outgoing edge to another node. A pink square marker is placed on this edge. Above the graph, the set $\{2, 3, 4\}$ is listed, and the equation $v_2 = v_1 + 1$ is written below the edge.



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

Problem 3.2.4

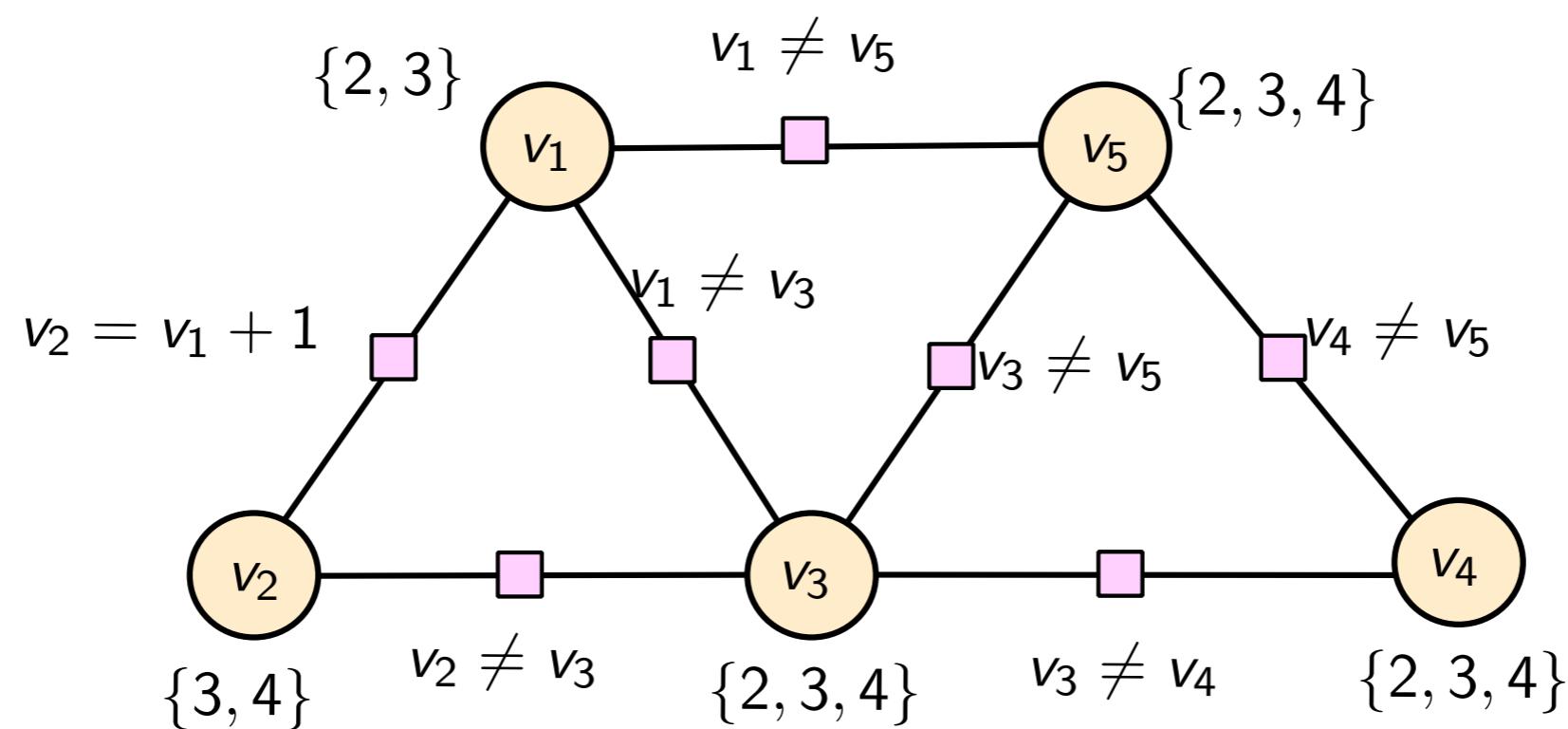
Consider the constraint satisfaction problem at its initial state. Is the CSP arc consistent? Is this a convenient initial condition if we plan to apply backtracking search? Describe the domains after the arc consistency algorithm has been applied to the CSP as a preprocessing step.



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

Problem 3.2.4

New, arc-consistent CSP:



The CSP is equivalent, however, we have reduced the sizes of some domains.

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 one randomly.

Value selection:

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

Inference:

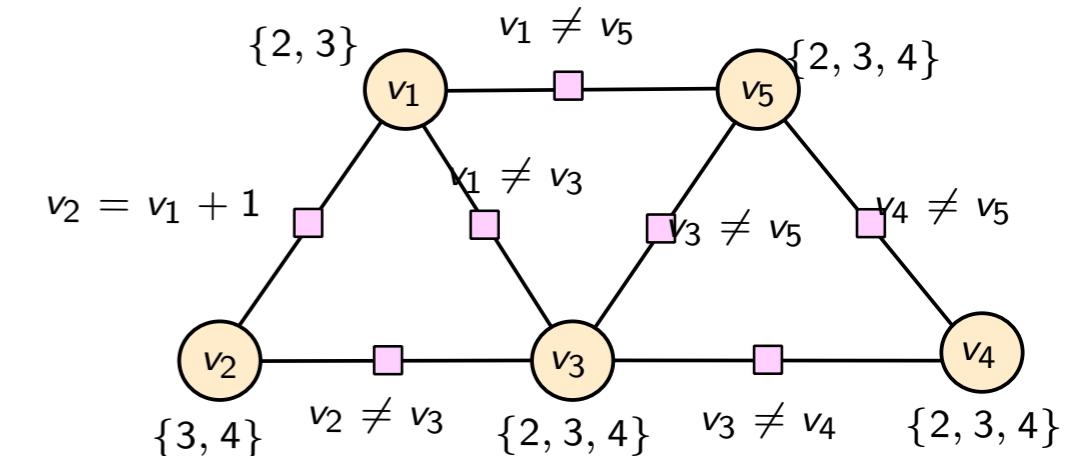
After each assignment: perform arc consistency algorithm. Backtrack if you find an inconsistency.

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

step	assign	current domains					degree					<i>backtrack to</i>
		v_1	v_2	v_3	v_4	v_5	v_1	v_2	v_3	v_4	v_5	

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

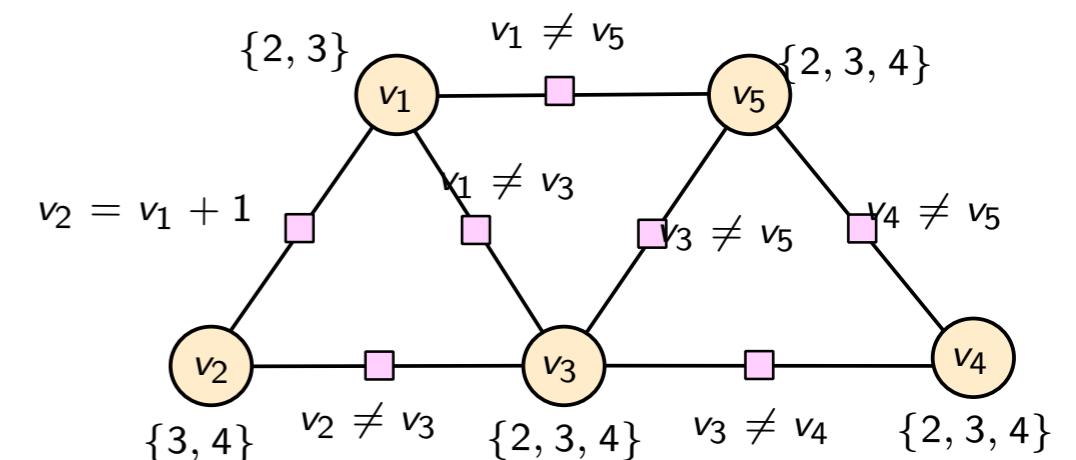


Problem 3.2.5: Backtracking Search after Preprocessing (AC-3)

Arc

Queue

Domain Change

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):

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):

8 iterations

2 backtracking steps

Arc Consistency Algorithm (3.2.3):

6 iterations

1 backtracking step

Arc Consistency Algorithm after preprocessing (3.2.5):

5 iterations

0 backtracking steps