

# Lecture 4

## Constraint Satisfaction Problems

Alice Gao

January 17, 2022

### Contents

<b>1</b>	<b>Learning Goals</b>	<b>2</b>
<b>2</b>	<b>Examples of CSP Problems</b>	<b>3</b>
<b>3</b>	<b>Formulating a CSP</b>	<b>7</b>
3.1	Why study CSPs separately? . . . . .	7
3.2	Components of a CSP . . . . .	8
3.3	Formulating 4-Queens Problem as a CSP . . . . .	8
<b>4</b>	<b>Backtracking Search</b>	<b>10</b>
4.1	Trace Backtracking Search on 4-Queens . . . . .	11
<b>5</b>	<b>The Arc Consistency Definition</b>	<b>15</b>
5.1	Motivation . . . . .	15
5.2	Handling Different Types of Constraints . . . . .	16
5.3	Notation for an Arc . . . . .	16
5.4	Arc Consistency Definition . . . . .	17
<b>6</b>	<b>The AC-3 Arc Consistency Algorithm</b>	<b>18</b>
6.1	Why do we need to add arcs back to S? . . . . .	19
6.2	Tracing the AC-3 Algorithm with $x_0 = 0$ . . . . .	20
6.3	Properties of the AC-3 algorithm . . . . .	28
<b>7</b>	<b>Practice Problems</b>	<b>29</b>

Modified and distributed by Blake VanBerlo, with permission from Alice Gao.

# 1 Learning Goals

By the end of the lecture, you should be able to

- Formulate a real-world problem as a constraint satisfaction problem.
- Trace the execution of the backtracking search algorithm.
- Verify whether a constraint is arc-consistent.
- Trace the execution of the AC-3 arc consistency algorithm.
- Trace the execution of the backtracking search algorithm with arc consistency.
- Trace the execution of the backtracking search algorithm with arc consistency and with heuristic for choosing variables and values.

## 2 Examples of CSP Problems

### Example:

Let's look at some real-world constraint satisfaction problems. Pascal Van Hentenryck is one of the world-leading experts on solving large real-world constraint satisfaction problems. He has worked on many different problems including disaster recovery and transportation planning.

Take evacuation planning as an example. Evacuation planning is critical in preparing and responding to natural and man-made disasters. In a controlled evacuation, agencies must instruct each endangered resident to follow a specific evacuation route at a given departure time.

There are two main challenges with evacuation planning. First, we must deploy enough resources to give instructions to the endangered population. Second, we must ensure that the endangered population complies with the evacuation orders.

Authorities still primarily rely on expert knowledge and simple heuristics to design and execute evacuation plans.

Existing evacuation approaches have several issues.

First, they violate a desirable constraint that all evacuees in a residential zone should preferably follow the same evacuation route. Several works addressed this issue by assigning a single evacuation path to each residential area.

Second, evacuees may not follow instructions if there are forks in the road. Latest work addressed this issue by constructing a forest of evacuation trees. In each tree, the root is at a safe place and the leaves are the residential areas.

Third, existing approaches allow preemption: That is, the evacuation can be interrupted and restarted arbitrarily. This is not desirable in practice since it will confuse both evacuees and emergency services and will be hard to enforce. Van Hentenryck's work generates non-preemptive evacuation schedules by using a constraint programming approach. They model the number of evacuees, the number of vehicles to be evacuated per time unit, the starting evacuation time and road capacities.

The constraint programming model was applied to a real-life evacuation case study in New South Wales, Australia. The region is susceptible to floods. A major spill would require evacuating 80,000 people. The model generated non-preemptive schedules that are close to the optimal schedules generated in prior work and yet are much more practical.

**Example:**

T	A	O	S		S	P	E	C	S		S	E	P	T
E	G	G	Y		A	L	A	A	P		I	L	I	A
T	H	R	E	E	P	E	N	N	Y		D	H	A	K
H	A	E		X	E	B	E	C		S	E	I	N	E
				V	O	L		D	H	O	W	S		
S	W	E	E	N	E	Y		A	N	A	T	M	A	N
P	R	Y	S		S	O	N		E	P	E	I	R	A
L	Y	R	I	C		N	O	S		S	P	A	E	R
I	L	I	C	E	S		R	U	C		P	O	C	K
T	Y	R	A	N	E	D		R	O	P	E	W	A	Y
				T	S	A	R	S		C	U	D		
B	A	G	I	E		O	I	N	K	S		T	S	K
O	L	E	O		S	W	E	E	P	S	T	A	K	E
Y	A	R	N		E	N	V	O	I		A	B	U	T
G	Y	M	S		I	D	E	N	T		G	U	G	A

Next, let's look at crossword puzzles. I presume that most of you have seen crossword puzzles.

Have you seen the movie Imitation Game? In the movie, Alan Turing first mentioned crossword puzzle during the interview at Bletchley Park. Let me replay some of the conversation for you:

Turing: I don't speak German.

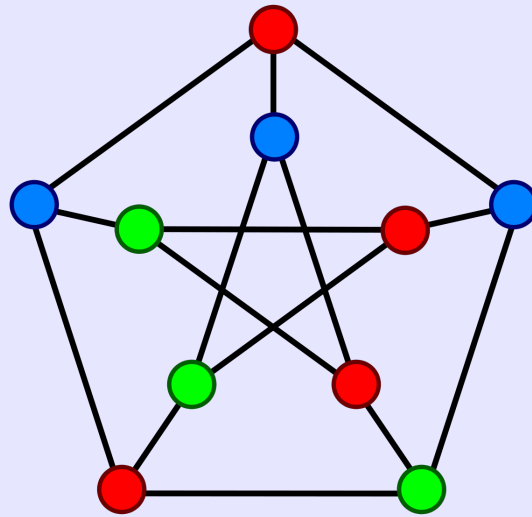
Denniston: How are you supposed to decrypt German communications?

Turing: Well, I'm really quite excellent at crossword puzzles. German codes are a puzzle. A game just like any other game. I like solving problems, Commander. And Enigma is the most difficult problem in the world.

Denniston: Enigma isn't difficult. It is impossible. Everyone thinks Enigma is unbreakable.

Turing: Good. Let me try, then we'll know for sure, won't we?

**Example:**



Next, we have the graph colouring problem. This is a well-known mathematical problem. We are given a graph and a set of colours. The question is: "Can we colour all the vertices in the graph such that no two adjacent vertices have the same colour?"

This question may look quite abstract and boring. However, it has many applications in the real world. Let me give you some examples:

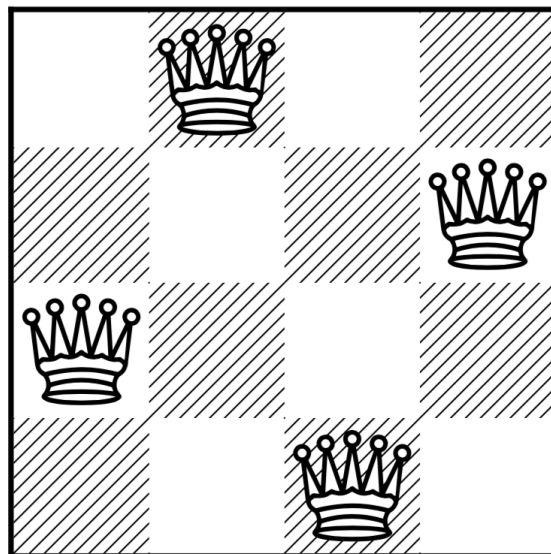
- Making a schedule or a time table.
- Assigning radio frequencies to the towers. We must assign different frequencies to towers in the same location.
- Register allocation in compiler optimization: We must assign a large number of target program variables onto a small number of CPU registers.
- Map coloring: In a geographical maps of countries or states, no two adjacent cities can be assigned the same color. There is a famous theorem for this problem: Four colors are sufficient to color any map.

**Example:**

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

The next example is a Sudoku puzzle. The goal is to complete all the empty squares with the numbers one to nine such that the numbers are all different in each row, in each column, and in each 3x3 grid. When I took the introduction to AI course at UBC, I had to write a program to solve Sudoku for one of the assignments. I struggled with writing the program quite a bit. But it was a lot of fun, and I learned a lot.

**Example:**



Finally, I have the four Queens problem. You should already be familiar with this problem. I will be using the 4-queens problem as a running example to discuss algorithms to solve a CSP.

## 3 Formulating a CSP

### 3.1 Why study CSPs separately?

We have spent quite a bit of time discussing search algorithms. Given this, why are we devoting a separate lecture for constraint satisfaction problems? How are CSPs different from other search problems?

Let's take a look at how we can solve 4-queens problem using a form of depth-first search. Start with an empty board and add queens to the board one by one. Once we've added four queens, the algorithm tests whether the state is a goal state or not. If it's not the goal state, we backtrack. We'll change the position of the last queen placed on the board. Whenever we have a full board, we will run the goal test again. This form of depth-first search is also called backtracking search.

Backtracking search treats each state as a black box. It does not know what is inside each state. Backtracking search can perform two tasks: (1) test whether a state is a goal, and (2) generate successors of a state.

Let me show you one reason why we may not be happy with backtracking search.

#### Example:

Q	Q		

Consider this partial state above. The two queens are in the same row. To us, it is clear that this state does not lead to a solution since the two queens already violate the row constraint. We should backtrack immediately and put the second queen in a different row.

Unfortunately, backtracking search does not know this. It only knows that this state is not a goal. Backtracking search needs to explore the entire sub-tree starting from this state to realize that this partial state does not lead to a solution.

For this example, it is very useful for the search algorithm to understand the internal structure of the state. In particular, if it knows and understands the positions of the queens, it may be able to prune more states complete the search much more quickly. Given this, when we are solving a CSP as a search problem, we will explicitly model the internal structure of each state, and we will discuss specialized algorithms that can solve CSPs efficiently.

### 3.2 Components of a CSP

Let's take a look at the components of a CSP. A CSP is one type of a search problem. Like any other search problem, we need to define

- The states
- The initial state.
- The goal states.
- The successor function.
- The cost function.

In addition, we will model the internal structure of each state. A state includes three parts, the variables, a domain for each variable, and a set of constraints. Each constraint involves one or more variables and specifies the combinations of allowable values for these variables. A solution to a CSP will include an assignment of values to all the variables such that this assignment satisfies all the constraints.

### 3.3 Formulating 4-Queens Problem as a CSP

Let's formulate the four queens problem as a CSP. I will focus on defining the components of a state.

How should we define the variables, the domains, and the constraint? Here is an idea that came to me first. We can define one variable for each queen. The variable stores the row and the column positions of the queen. The domain for each variable contains 16 possible positions for this queen. Our constraints are, no two queens can be in the same row, the same column, or the same diagonal.

#### Example:

- Variable:  $x_i$  represents the  $i$ -th queen.  $i \in \{0, 1, 2, 3\}$ .
- Domains:  $D_i = \{0, \dots, 15\}$ .
- Constraints: no two queens are in the same row, column, or diagonal.

This formulation works, but I will show you a better one. The second formulation keeps track of less information and has fewer constraints. We will make an assumption that there is exactly one queen in each column. Having assumed this, we no longer need to keep track of the column positions of the queens. We also don't have to worry about the column constraints.

Here are the details. I will define four variables,  $x_0$  to  $x_3$ . The subscript  $i$  refers to the column for each queen. For example,  $x_2$  is the queen placed in the third column from the left. The value of each variable is the row position of that queen. For example, if  $x_0$  has the value 0, the leftmost queen is in the top row. The domain of each variable contains the four



possible row positions. The constraints are, no two queens can be in the same row or in the same diagonal.

**Example:**

- Variables: There is exactly one queen in each column.  $x_0, x_1, x_2, x_3$  where  $x_i$  is the row position of the queen in column  $i$ , where  $i \in \{0, 1, 2, 3\}$ .
- Domains:  $D_{x_i} = \{0, 1, 2, 3\}$  for all  $x_i$ .
- Constraints: No two queens are in the same row or diagonal.

$$\forall_i \forall_j (i \neq j) \rightarrow ((x_i \neq x_j) \wedge (|x_i - x_j| \neq |i - j|))$$

The description of the constraints in words is quite clear. However, if you prefer, you can express the constraints as a logical formula. See the example above. The first part encodes the row constraint, and the second part encodes the diagonal constraint. If the queens are in the same diagonal, then the difference in their row positions must be equal to the difference in their column positions. We must require these two differences to be not equal. For example, to encode that the leftmost two queens cannot be in the same row and also cannot be in the same diagonal, we have  $x_0$  is not equal to  $x_1$  and the absolute difference between  $x_0$  and  $x_1$  is not 1.

## 4 Backtracking Search

In the previous section, I started formulating 4-queens as a CSP by defining the variables, the domains, and the constraints for each state. In this section, in order to solve 4-queens using backtracking search, let's finish complete the CSP formulation by defining the other components: initial state, goal states, and the successor function.

To apply backtracking search, we need to use an incremental formulation. An incremental formulation starts from an empty board and builds up the state step-by-step by adding one queen at each step.

First, let's complete our state definition. The variables correspond to the 4 queens. Since we use an incremental formulation, let's assume that we are adding the queens one by one from left to right. When there are  $k$  queens on the board, the queens are in the leftmost  $k$  columns. Let's enforce the constraints at all times and assume that the  $k$  queens on the board do not attack each other directly or indirectly. We'll denote each state by four characters. If the queen is on the board, the character is an integer indicating the row position of the queen. If the queen is not on the board, the character is a blank.

The initial state is the empty state, denoted by four blanks.

There are several goal states. Any goal state must have all four queens on the board, satisfying all the constraints.

Finally, how do we generate a successor? We can generate the successor by adding a queen in the first empty column from the left, making sure that the new queen does not attack any existing queen.

For example, what are the successors of the state 0 \_ \_ ? The second queen cannot be in row 0 due to the row constraint. The second queen cannot be in row 1 due to the diagonal constraint. Thus, the second queen can be 2 or 3. The two successors are 0 2 \_ \_ and 0 3 \_ \_.

We made two simplifying assumptions in our formulation. The first assumption is that "we always add queens from left to right." This assumption helps eliminate duplicate states. If we can add the queens in any order, then adding two queens in different orders may generate the same state multiple times.

The second assumption is that every state must satisfy all the constraints. This assumption comes with a trade-off. If we make this assumption, we need to eliminate boards violating any constraint when generating successors. As a result, there are fewer successors and the search tree is smaller. On the other hand, if we don't make this assumption, generating successors requires less work, but we must prune states that violate any constraints in the search tree.

We now have a complete incremental CSP formulation for the 4-queens problem.

**Example:**

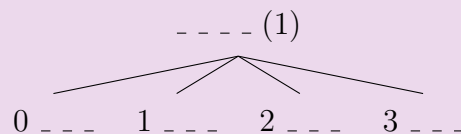
- State: one queen per column in the leftmost  $k$  columns with no pair of queens attacking each other.
- Initial state: no queens on the board.
- Goal state: 4 queens on the board. No pair of queens are attacking each other.
- Successor function: add a queen to the leftmost empty column such that it is not attacked by any other existing queen.

## 4.1 Trace Backtracking Search on 4-Queens

Let's trace backtracking search on the 4-queens problem.

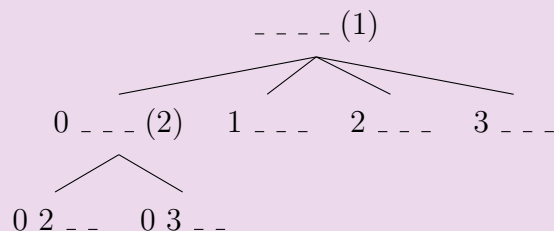
### Solution:

The initial state is the empty state. The leftmost queen has four possible positions. Let's think of each state as a four-digit number and we will expand the states in lexicographical order.



	$x_0$	$x_1$	$x_2$	$x_3$
0				
1				
2				
3				

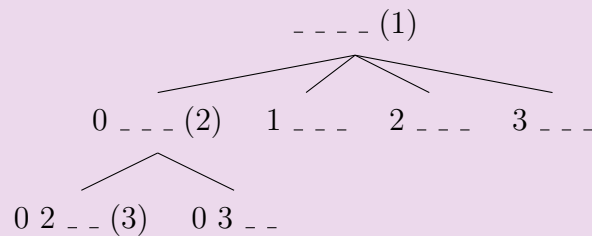
Consider the leftmost state 0 \_ \_ . The second queen has two possible positions, 2 and 3. There are two successors.



	$x_0$	$x_1$	$x_2$	$x_3$
0	Q	x	x	x
1	x	x		
2	x		x	
3	x			x

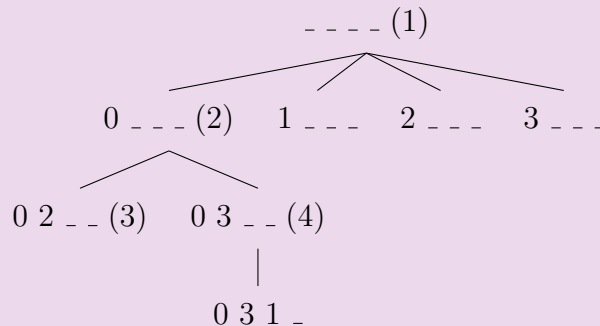
Next, let's expand the leftmost state 0 2 \_ . Unfortunately, there are no possible

positions for the third queen. This state has no successor. We need to backtrack.



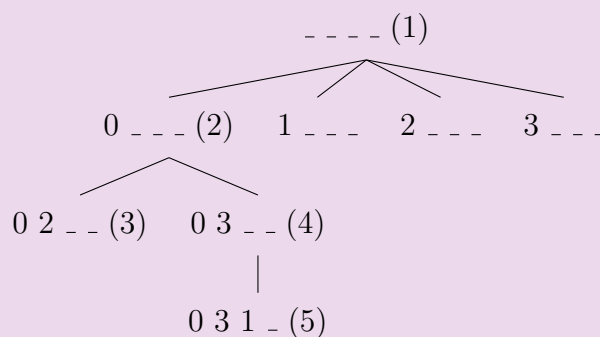
	$x_0$	$x_1$	$x_2$	$x_3$
0	Q	x	x	x
1	x	x	x	
2	x	Q	x	x
3	x	x	x	x

Coming back to the parent state 0 -- -. Let's expand the next successor, 0 3 --. The third queen has one possible position, row 1. The only successor is 0 3 1 --.



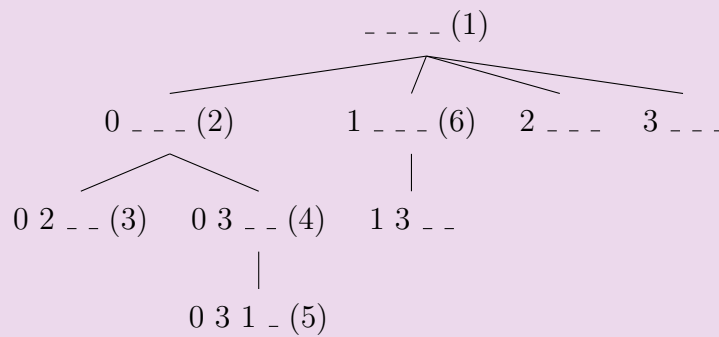
	$x_0$	$x_1$	$x_2$	$x_3$
0	Q	x	x	x
1	x	x	Q	x
2	x	x	x	x
3	x	Q	x	x

Let's expand the state 0 3 1 --. Unfortunately, there are no possible positions for the last queen. We need to backtrack.



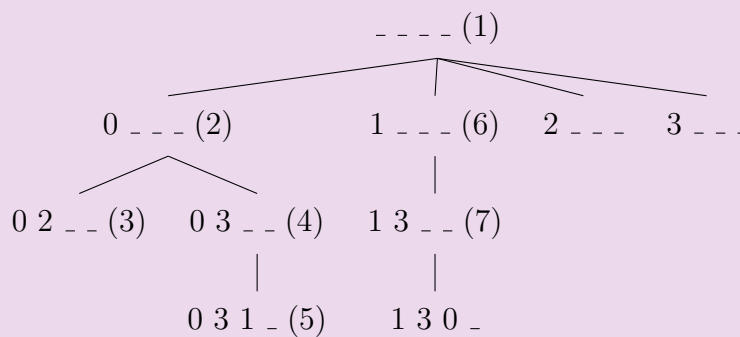
	$x_0$	$x_1$	$x_2$	$x_3$
0	Q	x	x	x
1	x	x	Q	x
2	x	x	x	x
3	x	Q	x	x

This time, backtracking takes us all the way back to the initial state. Let's expand the second successor: 1 -- -. This state has one successor, 1 3 --.



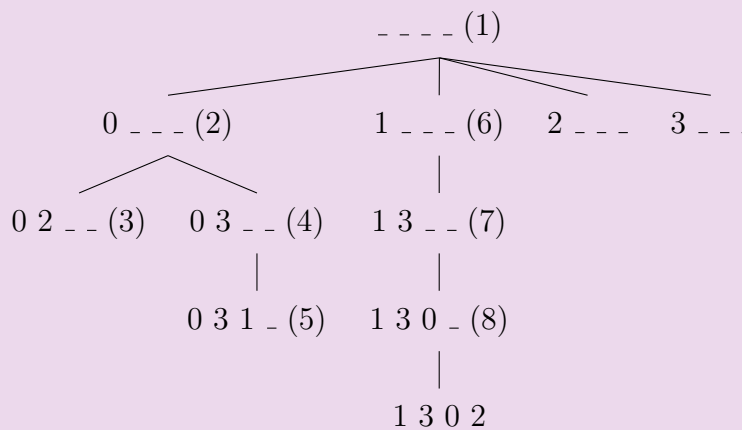
	$x_0$	$x_1$	$x_2$	$x_3$
0	x	x		
1	Q	x	x	x
2	x	x	x	
3	x	Q	x	x

Let's expand this state 1 3 \_ -. This state has one successor, 1 3 0 \_.



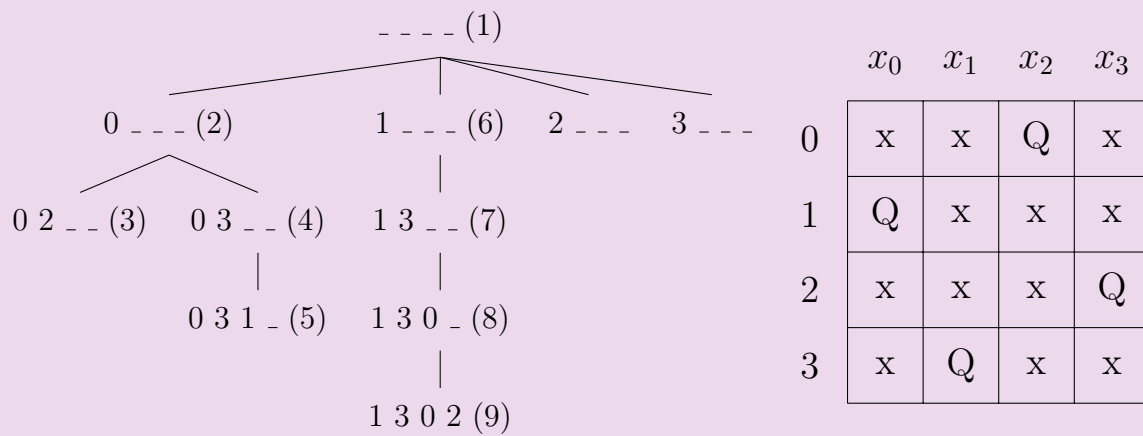
	$x_0$	$x_1$	$x_2$	$x_3$
0	x	x	Q	x
1	Q	x	x	x
2	x	x	x	
3	x	Q	x	x

Let's expand this state 1 3 0 -. It has one successor 1 3 0 2.



	$x_0$	$x_1$	$x_2$	$x_3$
0	x	x	Q	x
1	Q	x	x	x
2	x	x	x	Q
3	x	Q	x	x

Next, let's expand the state 1 3 0 2. This is a goal state and the algorithm terminates.



Let's make two observations. First, backtracking search is similar to depth-first search. We dive deep into the tree and backtrack when we reach a dead end. Second, our search is quite efficient. Enforcing the constraints allows us to generate fewer successors. For example, we never considered the state 0 0 \_ \_ since it is not a valid successor of 0 \_ \_ \_.

## 5 The Arc Consistency Definition

### 5.1 Motivation

First, why do we want to use arc consistency? It provides another way for us to solve a CSP.

In the previous section, I solved the 4-queens problem using backtracking search. In the leftmost sub-tree, we started by putting the first queen in row 0 and later on realized that this assignment does not lead to a solution. It turns out that, if we make use of arc consistency, we can detect this dead end without expanding the leftmost sub-tree at all.

Let me illustrate the idea of arc consistency using an example.

**Example:** Suppose that we put the first queen  $x_0$  in row 0. After this, due to the row and diagonal constraints, queen  $x_1$  cannot be 0 or 1. What if we put queen  $x_1$  in row 2? How can we figure out that this assignment does not lead to a solution?

	$x_0$	$x_1$	$x_2$	$x_3$
0	Q	X		
1		X		
2		Q		
3				

Since queen  $x_0$  is in row 0, we cannot put any queen in the top row. We also cannot put any queen in the diagonal containing  $x_0$ .

	$x_0$	$x_1$	$x_2$	$x_3$
0	Q	X	X	X
1		X		
2		Q	X	
3				X

Similarly, since queen  $x_1$  is in row 2, we cannot put any queen in row 2, and we cannot put any queen in the two diagonals containing  $x_2$ .

	$x_0$	$x_1$	$x_2$	$x_3$
0	Q	X	X	X
1		X	X	
2	X	Q	X	X
3			X	X

At this point, all possible positions for queen  $x_2$  have been eliminated. Therefore, the assignment  $x_0 = 0$  and  $x_1 = 2$  does not lead to a solution.

This example illustrates the idea behind arc-consistency. By examining each constraint, we can eliminate certain values that cannot satisfy the constraint from some domains. Reducing the domains this way can help us get closer to finding a solution or proving that a solution does not exist.

## 5.2 Handling Different Types of Constraints

I will introduce an arc-consistency algorithm which handles binary constraints only. We will categorize each constraint based on the number of variables in the constraint. A unary constraint involves one variable. A binary constraint involves two variables. There are constraints involving 3 or more variables.

Restricting to binary constraints does not limit the applicability of our algorithm. In fact, this algorithm can solve any CSP. How does it handle other types of constraints? Handling a unary constraint is straightforward. A unary constraint says whether a variable can take a value or not. If a constraint does not allow a value, we can simply remove it from the variable's domain. What about a constraint involving 3 or more variables? It turns out that we can convert the constraint into an equivalent set of binary constraints by defining additional variables. How to convert these constraints is beyond the scope of this course. You can look it up if you are interested.

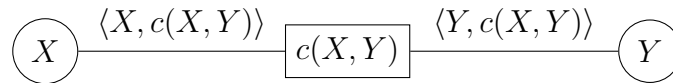
## 5.3 Notation for an Arc

Before I introduce the arc-consistency definition, let me introduce some notation for defining an arc.

Let  $X$  and  $Y$  be two variables. Let  $c(X, Y)$  be a binary constraint. Depending on the meaning of the constraint,  $c(X, Y)$  and  $c(Y, X)$  may or may not represent the same constraint. For example, if the constraint is  $X + Y = 5$ , then  $c(X, Y)$  and  $c(Y, X)$  have the same meaning. If the constraint is  $X$  is divisible by  $Y$ , then  $c(X, Y)$  and  $c(Y, X)$  have different meanings.

Each constraint has two arcs associated with it. Take a look at the picture.





Let's look at the left arc:  $\langle X, c(X, Y) \rangle$ . The arc is denoted by a tuple in angle brackets. The second element in the tuple is the constraint. The first element in the tuple is the primary variable in the arc.

The primary variable tells us which arc we are looking at. For our example, if the primary variable is  $X$ , we are looking at the one on the left. If the primary variable is  $Y$ , we are looking at the one on the right.

## 5.4 Arc Consistency Definition

Let's look at the arc consistency definition.

**Definition** (Arc Consistency). An arc  $\langle X, c(X, Y) \rangle$  is consistent if and only if for every value  $v$  in  $D_X$ , there is a value  $w$  in  $D_Y$  such that  $(v, w)$  satisfies the constraint  $c(X, Y)$ .

There are two variables  $X$  and  $Y$  with domains  $D_X$  and  $D_Y$ .  $c(X, Y)$  is binary constraint. Consider the arc with  $X$  as the primary variable. This arc is consistent if and only if, for every value  $v$  in the domain of  $X$ , there is a value  $w$  in the domain of  $Y$ , such that  $v$  and  $w$  satisfy the constraint  $c$ .

To help you understand the definition, let's translate the definition into a predicate logic formula. Stop reading and write down the logical formula yourself. Then, keep reading for the answer.

$$\forall v \in D_X, \exists w \in D_Y, (v, w) \text{ satisfies } c(X, Y)$$

In predicate logic, the definition becomes the following. For all  $v$  in the domain of  $X$ , there exists  $w$  in the domain of  $Y$ , such that  $v$  and  $w$  satisfy the constraint  $c$ . We have mixed quantifiers here. The first one is universal and the second one is existential. When we verify the statement, we need to consider the quantifiers in order. First, we need to pick a value  $v$  in the domain of  $X$ . Then, we need to find a value  $w$  in the domain of  $Y$  such that the  $v$  and  $w$  satisfy the constraint.

Let's look at an example of applying the arc consistency definition.

**Problem:** Consider the constraint  $X < Y$ . Let  $D_X = \{1, 2\}$  and  $D_Y = \{1, 2\}$ . Is the arc  $\langle X, X < Y \rangle$  consistent?

- (A) Yes.
- (B) No.
- (C) I don't know.

**Solution:** The correct answer is (B). The arc is NOT consistent.

To verify the definition, we start by picking any value of  $X$  and try to find a value of  $Y$  such that the two values satisfy the constraint.

If  $X = 1$ , then  $Y = 2$  satisfy the constraint.

If  $X = 2$ , then no value of  $Y$  satisfies the constraint. Since we cannot find a corresponding value of  $Y$  for  $X = 2$ , this arc is not consistent. We can go ahead and remove 2 from the domain of  $X$ .

In general, if an arc is not consistent, we can remove at least one value from the primary variable's domain. Doing this will not rule out any solution. Reducing the domains is helpful as it takes us closer to finding a solution or showing that a solution does not exist.

## 6 The AC-3 Arc Consistency Algorithm

In the previous section, I introduced the Arc Consistency definition. The AC-3 algorithm uses the arc-consistency definition to reduce the variables' domains and bring us closer to solving the CSP.

The AC-3 algorithm was proposed in 1977 by Alan Mackworth, who is a professor emeritus in Computer Science at UBC. He's also one of the two authors in the Poole and Mackworth textbook. Also, you should know that Peter Van Beek, a CS professor at UW worked extensively on constraint satisfaction problems. The Russell and Norvig textbook has many citations of research by Professor Van Beek. I think it's very exciting that we learning concepts that are developed by our professors and colleagues.

---

### Algorithm 1 The AC-3 Algorithm

---

```

1: put every arc in the set  $S$ .
2: while  $S$  is not empty do
3:   select and remove  $\langle X, c(X, Y) \rangle$  from  $S$ 
4:   remove every value in  $D_X$  that doesn't have a value in  $D_Y$  that satisfies the constraint
      $c(X, Y)$ 
5:   if  $D_X$  was reduced then
6:     if  $D_X$  is empty then return false
7:     for every  $Z \neq Y$ , add  $\langle Z, c'(Z, X) \rangle$  to  $S$ 
return true

```

---

The goal of the AC-3 algorithm is to make every arc consistent by reducing the variables' domains.

The algorithm starts by putting every arc into a set called  $S$ . Remember that each binary constraint has two arcs associated with it. We need to put both arcs into  $S$ .

Next, we'll go into a loop. While the set  $S$  is not empty, select and remove one arc from the set. The order of removing arcs is not important. Any order will lead us to the same solution.

Suppose that the arc we removed is between  $X$  and  $Y$ , where  $X$  is the primary variable. Next, check whether this arc is consistent. If the arc is consistent, we can go back to the beginning of the loop and select another arc to remove.

If the arc is not consistent, we will reduce the domain of the primary variable ( $X$ 's domain) until the arc becomes consistent. After reducing  $X$ 's domain, if the domain becomes empty, we know that the problem has no solution and the algorithm can terminate.

If  $X$ 's domain is not empty, we may need to add one or more arcs back to  $S$ . At this step, consider any arc where  $X$  is the secondary variable and  $Y$  is not the primary variable. If any such arc is not in  $S$ , we will add the arc to  $S$ .

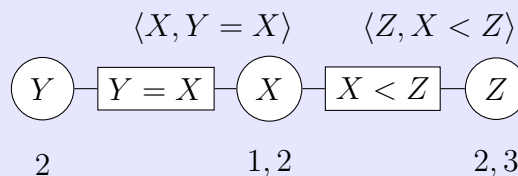
## 6.1 Why do we need to add arcs back to $S$ ?

The last few lines in the algorithm deserves some further discussion. In particular, after reducing a variable's domain, why do we need to add these arcs back into the set?

The reason is that, reducing a variable's domain may cause a previously consistent arc to become inconsistent. If this is a possibility, we need to add the arc back to  $S$  so that we will check it again.

Let's look at an example.

**Example:** The diagram shows three variables:  $Y$ ,  $X$ , and  $Z$ . There are two binary constraints:  $Y = X$ , and  $X < Z$ . Consider two arcs,  $\langle X, Y = X \rangle$ , and  $\langle Z, X < Z \rangle$ .

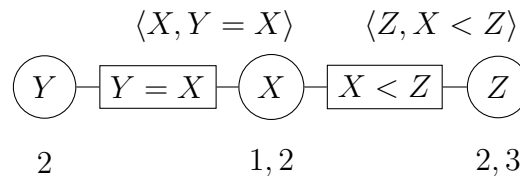


Right now, the right arc  $\langle Z, X < Z \rangle$  is consistent, but the left arc  $\langle X, Y = X \rangle$  is NOT consistent. Let's verify this quickly. For the right arc  $\langle Z, X < Z \rangle$ , if  $Z = 2$ , we can choose  $X = 1$  and  $X$  is less than  $Z$ . If  $Z = 3$ , we can choose  $X = 2$  and  $X$  is less than  $Z$ . Therefore, the arc is consistent. For the left arc  $\langle X, Y = X \rangle$ , if  $X$  is 2, we can choose  $Y = 2$ . If  $X = 1$ , no value of  $Y$  can satisfy the constraint.

To make the left arc consistent, we need to remove 1 from  $X$ 's domain. After removing 1 from  $X$ 's domain, the left arc  $\langle X, Y = X \rangle$  becomes consistent. What about the right arc  $\langle Z, X < Z \rangle$ ? The right arc becomes inconsistent because of this change. Let's verify it. If  $Z = 3$ , we can choose  $X = 2$  and  $X$  is less than  $Z$ . If  $Z = 2$ , no value of  $X$  can satisfy the constraint  $X < Z$ .

Before we reduced  $X$ 's domain, the right arc  $\langle Z, X < Z \rangle$  was consistent. After reducing  $X$ 's domain, the right arc  $\langle Z, X < Z \rangle$  becomes inconsistent. In this case, we need to add  $\langle Z, X < Z \rangle$  back to  $S$  so that we make it consistent by reducing  $Z$ 's domain.

Finally, there is an edge case to consider. Let's look at the diagram again.



To make  $\langle X, Y = X \rangle$  consistent, we need to reduce  $X$ 's domain. After this change, we do not need to add the arc  $\langle Y, Y = X \rangle$  to  $S$ . In other words, we do not need to add back the other arc for the same constraint. I will leave it as an exercise for you. Below is the exact statement that you will want to prove.

**Problem:** Consider two variables  $X$  and  $Y$ , and a binary constraint  $c(X, Y)$ . Assume that  $\langle X, c(X, Y) \rangle$  is NOT consistent and  $\langle Y, c(X, Y) \rangle$  is consistent. Assume that after removing one value from  $X$ 's domain,  $\langle X, c(X, Y) \rangle$  becomes consistent. Prove that after reducing  $X$ 's domain,  $\langle Y, c(X, Y) \rangle$  remains consistent.

## 6.2 Tracing the AC-3 Algorithm with $x_0 = 0$

When I solved the 4-queens problems using backtracking search. I showed you that, assigning  $x_0 = 0$  does not lead to a solution. You can see this on the leftmost branch of the search tree. In this example, I will show you that, we can arrive at the same conclusion by running the AC-3 algorithm.

See the constraint graph for the 4-queens problem below. Let's start by keeping only 0 in the domain of  $x_0$ . This means that we fix  $x_0$  to be 0.

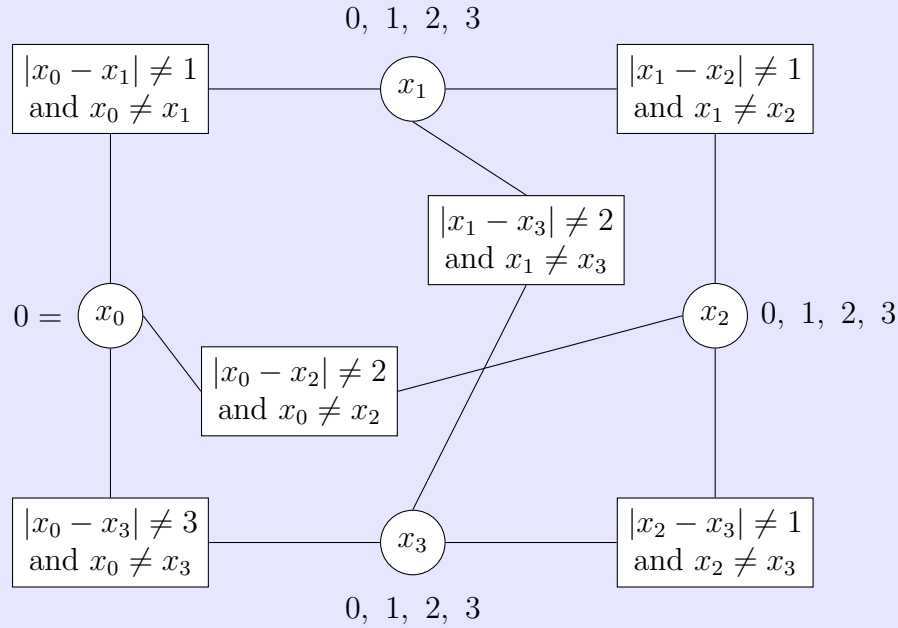
There are 6 constraints, and there are 2 arcs per constraints. Therefore, at the start of the AC-3 algorithm, we will add 12 arcs to the set  $S$ .

We will describe each step in words and also using a picture. The domain of each variable  $x_i$  is shown beside the corresponding node. We will indicate arcs in the set  $S$  with solid lines, and arcs not in the set  $S$  with dashed lines. The changes at each step are indicated by corresponding numbers.

### Example:

To start, the set  $S$  has 12 arcs:  $\langle x_0, c(x_0, x_1) \rangle$   $\langle x_1, c(x_0, x_1) \rangle$   $\langle x_0, c(x_0, x_2) \rangle$   $\langle x_2, c(x_0, x_2) \rangle$   $\langle x_0, c(x_0, x_3) \rangle$   $\langle x_3, c(x_0, x_3) \rangle$   $\langle x_1, c(x_1, x_2) \rangle$   $\langle x_2, c(x_1, x_2) \rangle$   $\langle x_1, c(x_1, x_3) \rangle$   $\langle x_3, c(x_1, x_3) \rangle$   $\langle x_2, c(x_2, x_3) \rangle$   $\langle x_3, c(x_2, x_3) \rangle$

The domains of the variables are:  $x_0 \in \{0\}, x_1 \in \{0, 1, 2, 3\}, x_2 \in \{0, 1, 2, 3\}, x_3 \in \{0, 1, 2, 3\}$ .



**Step 1: Remove  $\langle x_1, c(x_0, x_1) \rangle$  from the set  $S$ .**

$\langle x_1, c(x_0, x_1) \rangle$  is NOT arc-consistent. We need to remove 0 and 1 from the domain of  $x_1$ .

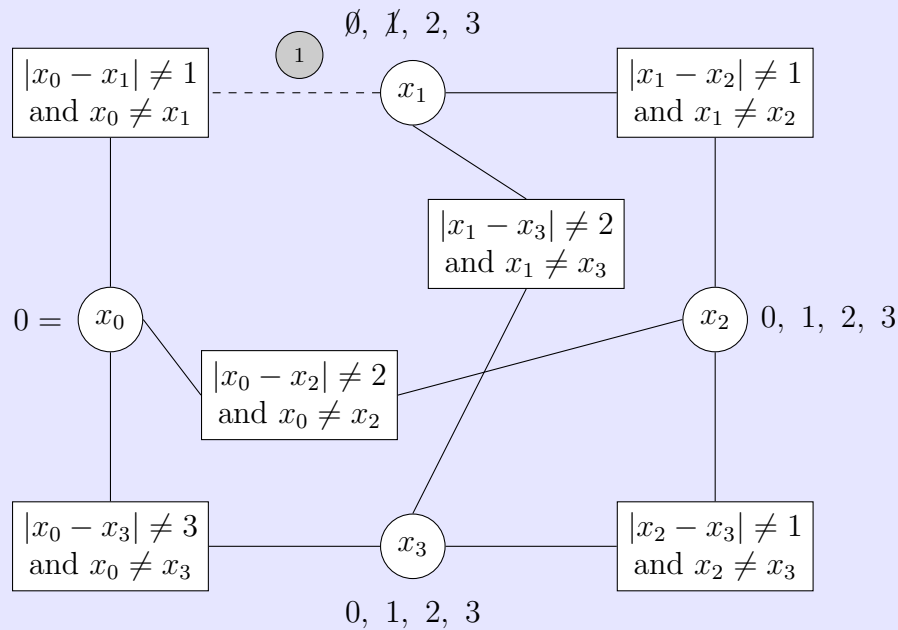
After reducing the domain of  $x_1$ , we possibly want to add back the following arcs:  $\langle x_0, c(x_0, x_1) \rangle$   $\langle x_2, c(x_1, x_2) \rangle$   $\langle x_3, c(x_1, x_3) \rangle$ .

Due to the special case in the algorithm, we do not need to add back the arc for the same constraint where the other variable is the primary variable. Thus, we do not want to add back  $\langle x_0, c(x_0, x_1) \rangle$ .

We should add the other 2 arcs back to  $S$ . However, both are already in  $S$  and  $S$  is a set with no duplicates. Thus, we do not need to add them again.

The set  $S$  becomes:  $\langle x_0, c(x_0, x_1) \rangle$   $\langle x_0, c(x_0, x_2) \rangle$   $\langle x_2, c(x_0, x_2) \rangle$   $\langle x_0, c(x_0, x_3) \rangle$   $\langle x_3, c(x_0, x_3) \rangle$   $\langle x_1, c(x_1, x_2) \rangle$   $\langle x_2, c(x_1, x_2) \rangle$   $\langle x_1, c(x_1, x_3) \rangle$   $\langle x_3, c(x_1, x_3) \rangle$   $\langle x_2, c(x_2, x_3) \rangle$   $\langle x_3, c(x_2, x_3) \rangle$

The domains of the variables become:  $x_0 \in \{0\}, x_1 \in \{2, 3\}, x_2 \in \{0, 1, 2, 3\}, x_3 \in \{0, 1, 2, 3\}$ .



**Step 2: Remove  $\langle x_2, c(x_0, x_2) \rangle$  from the set  $S$ .**

$\langle x_2, c(x_0, x_2) \rangle$  is NOT arc-consistent. We need to remove 0 and 2 from the domain of  $x_2$ .

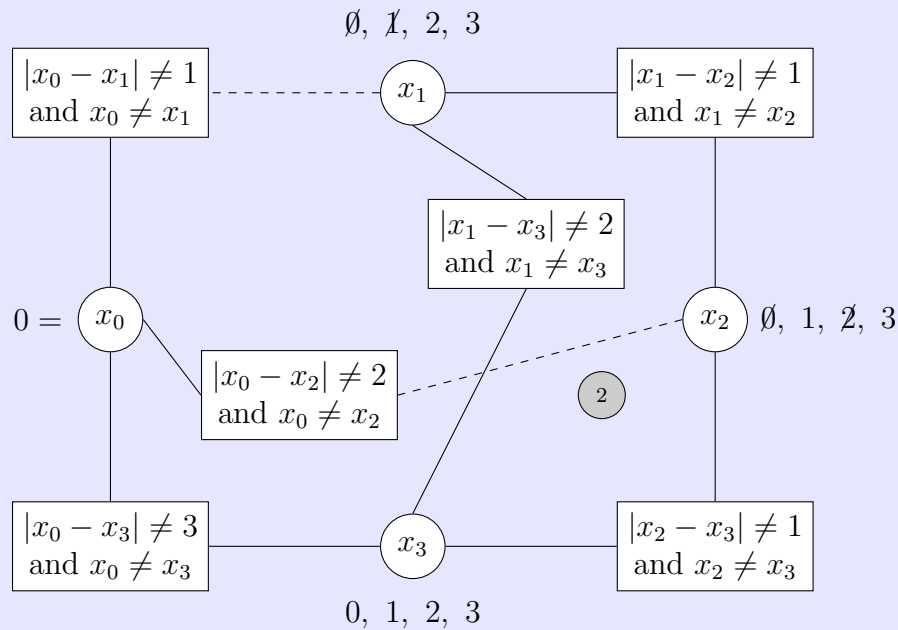
After reducing the domain of  $x_1$ , we possibly want to add back the following arcs:  
 $\langle x_0, c(x_0, x_2) \rangle$   $\langle x_1, c(x_1, x_2) \rangle$   $\langle x_3, c(x_2, x_3) \rangle$ .

Due to the special case in the algorithm, we do not need to add back the arc for the same constraint where the other variable is the primary variable. Thus, we do not want to add back  $\langle x_0, c(x_0, x_2) \rangle$ .

We should add the other 2 arcs back to  $S$ . However, both are already in  $S$  and  $S$  is a set with no duplicates. Therefore, we do not need add any arc back to  $S$ .

The set  $S$  becomes:  $\langle x_0, c(x_0, x_1) \rangle$   $\langle x_0, c(x_0, x_2) \rangle$   $\langle x_0, c(x_0, x_3) \rangle$   $\langle x_3, c(x_0, x_3) \rangle$   
 $\langle x_1, c(x_1, x_2) \rangle$   $\langle x_2, c(x_1, x_2) \rangle$   $\langle x_1, c(x_1, x_3) \rangle$   $\langle x_3, c(x_1, x_3) \rangle$   $\langle x_2, c(x_2, x_3) \rangle$   $\langle x_3, c(x_2, x_3) \rangle$

The domains of the variables become:  $x_0 \in \{0\}$ ,  $x_1 \in \{2, 3\}$ ,  $x_2 \in \{1, 3\}$ ,  $x_3 \in \{0, 1, 2, 3\}$ .



**Step 3: Remove  $\langle x_3, c(x_0, x_3) \rangle$  from the set  $S$ .**

$\langle x_3, c(x_0, x_3) \rangle$  is NOT arc-consistent. We need to remove 0 and 3 from the domain of  $x_3$ .

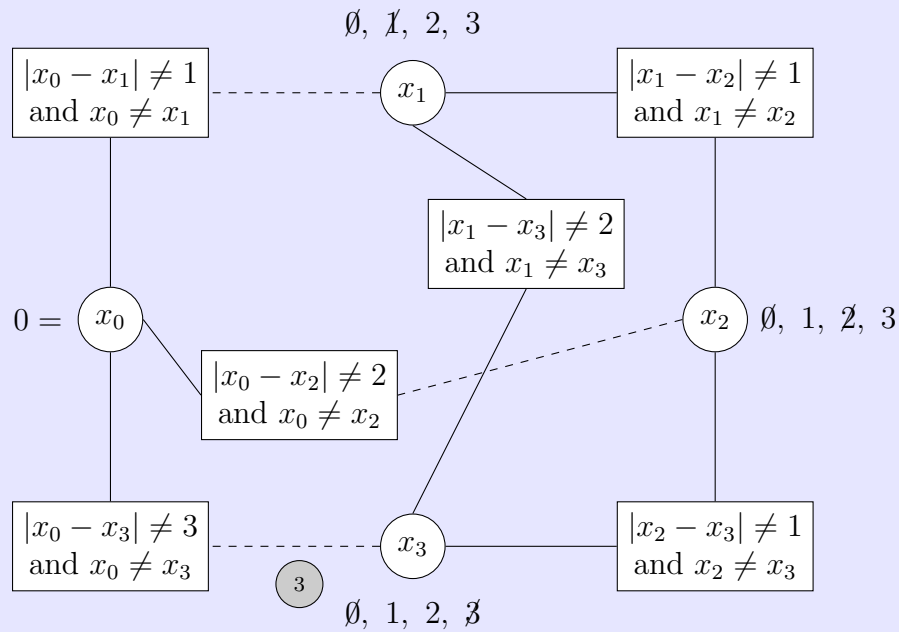
After reducing the domain of  $x_1$ , we possibly want to add back the following arcs:  $\langle x_0, c(x_0, x_3) \rangle$   $\langle x_1, c(x_1, x_3) \rangle$   $\langle x_2, c(x_2, x_3) \rangle$ .

Due to the special case in the algorithm, we do not need to add back the arc for the same constraint where the other variable is the primary variable. Thus, we do not want to add back  $\langle x_0, c(x_0, x_3) \rangle$ .

We should add the other 2 arcs back to  $S$ . However, both are already in  $S$  and  $S$  is a set with no duplicates. Therefore, we do not need add any arc back to  $S$ .

The set  $S$  becomes:  $\langle x_0, c(x_0, x_1) \rangle$   $\langle x_0, c(x_0, x_2) \rangle$   $\langle x_0, c(x_0, x_3) \rangle$   $\langle x_1, c(x_1, x_2) \rangle$   $\langle x_2, c(x_1, x_2) \rangle$   $\langle x_1, c(x_1, x_3) \rangle$   $\langle x_3, c(x_1, x_3) \rangle$   $\langle x_2, c(x_2, x_3) \rangle$   $\langle x_3, c(x_2, x_3) \rangle$

The domains of the variables become:  $x_0 \in \{0\}$ ,  $x_1 \in \{2, 3\}$ ,  $x_2 \in \{1, 3\}$ ,  $x_3 \in \{1, 2\}$ .



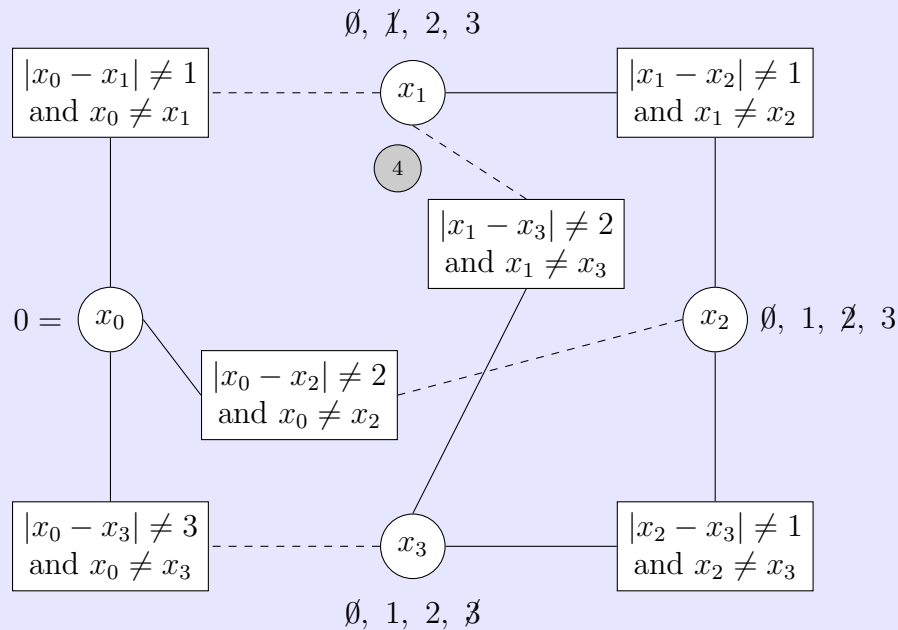
**Step 4: Remove  $\langle x_1, c(x_1, x_3) \rangle$  from the set  $S$ .**

$\langle x_1, c(x_1, x_3) \rangle$  is arc-consistent. Thus, we do not reduce the domain of  $x_1$ , and we do not add any arc back to  $S$ .

The set  $S$  becomes:  $\langle x_0, c(x_0, x_1) \rangle$   $\langle x_0, c(x_0, x_2) \rangle$   $\langle x_0, c(x_0, x_3) \rangle$   $\langle x_1, c(x_1, x_2) \rangle$   
 $\langle x_2, c(x_1, x_2) \rangle$   $\langle x_3, c(x_1, x_3) \rangle$   $\langle x_2, c(x_2, x_3) \rangle$   $\langle x_3, c(x_2, x_3) \rangle$

The domains of the variables become:  $x_0 \in \{0\}$ ,  $x_1 \in \{2, 3\}$ ,  $x_2 \in \{1, 3\}$ ,  $x_3 \in \{1, 2\}$ .





**Step 5: Remove  $\langle x_2, c(x_2, x_3) \rangle$  from the set  $S$ .**

$\langle x_2, c(x_2, x_3) \rangle$  is NOT arc-consistent. We need to remove 1 from the domain of  $x_2$ .

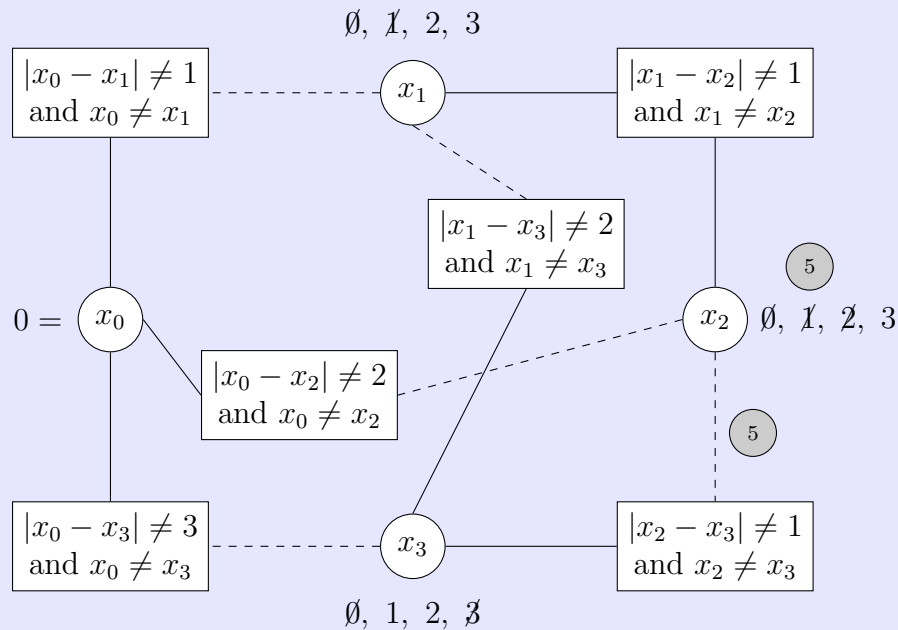
After reducing the domain of  $x_2$ , we possibly want to add back the following arcs:  $\langle x_0, c(x_0, x_2) \rangle$   $\langle x_1, c(x_1, x_2) \rangle$   $\langle x_3, c(x_3, x_2) \rangle$ . These are the arcs where  $x_2$  is the secondary variable ( $x_2$  is not the first element in the tuple).

We do not want to add back  $\langle x_3, c(x_3, x_2) \rangle$  because there is no need to add back the same constraint for the other variable. (There is a proof for this.)

Therefore, we will add  $\langle x_0, c(x_0, x_2) \rangle$  and  $\langle x_1, c(x_1, x_2) \rangle$  back to  $S$ . These arcs are already in  $S$  and  $S$  is a set with no duplicates. Thus,  $S$  does not change.

The set  $S$  becomes:  $\langle x_0, c(x_0, x_1) \rangle$   $\langle x_0, c(x_0, x_2) \rangle$   $\langle x_0, c(x_0, x_3) \rangle$   $\langle x_1, c(x_1, x_2) \rangle$   $\langle x_2, c(x_1, x_2) \rangle$   $\langle x_3, c(x_1, x_3) \rangle$   $\langle x_3, c(x_2, x_3) \rangle$

The domains of the variables become:  $x_0 \in \{0\}$ ,  $x_1 \in \{2, 3\}$ ,  $x_2 \in \{3\}$ ,  $x_3 \in \{1, 2\}$ .



**Step 6: Remove  $\langle x_3, c(x_2, x_3) \rangle$  from the set  $S$ .**

$\langle x_3, c(x_2, x_3) \rangle$  is NOT arc-consistent. We need to remove 2 from the domain of  $x_3$ .

After removing 2 from the domain of  $x_3$ , we possibly want to add back the following arcs:  $\langle x_0, c(x_0, x_3) \rangle$   $\langle x_1, c(x_1, x_3) \rangle$   $\langle x_2, c(x_2, x_3) \rangle$ .

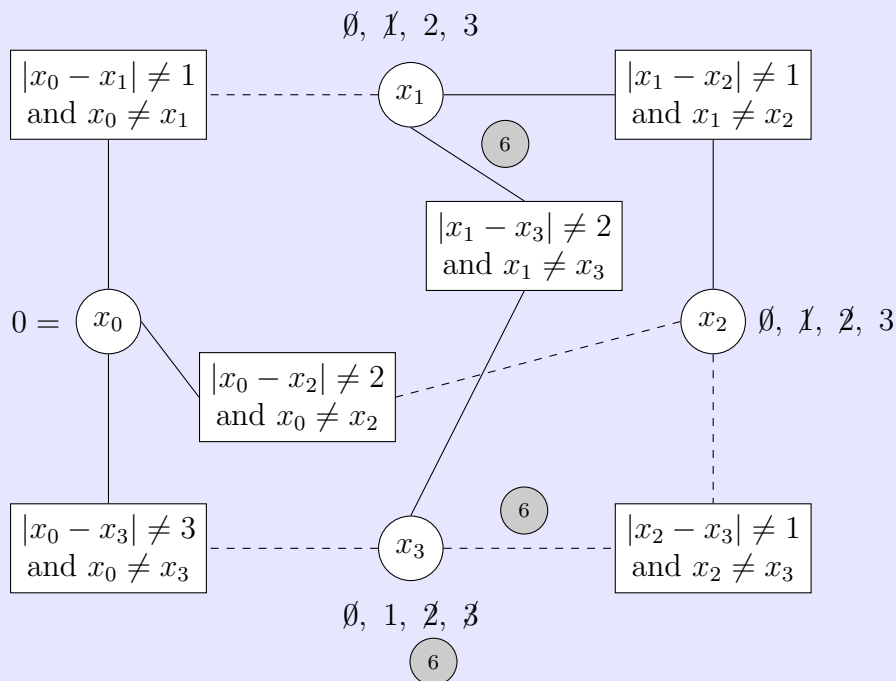
We do not want to add back  $\langle x_0, c(x_0, x_3) \rangle$  to the set because it is already in  $S$ .

We do not want to add back  $\langle x_2, c(x_2, x_3) \rangle$  because there is no need to add back the same constraint for the other variable. (There is a proof for this.)

Therefore, we will add  $\langle x_1, c(x_1, x_3) \rangle$  back to  $S$ .  $S$  changes because it does not have this arc in it.

The set  $S$  becomes:  $\langle x_0, c(x_0, x_1) \rangle$   $\langle x_0, c(x_0, x_2) \rangle$   $\langle x_0, c(x_0, x_3) \rangle$   $\langle x_1, c(x_1, x_2) \rangle$   $\langle x_2, c(x_1, x_2) \rangle$   $\langle x_3, c(x_1, x_3) \rangle$   $\langle x_1, c(x_1, x_3) \rangle$

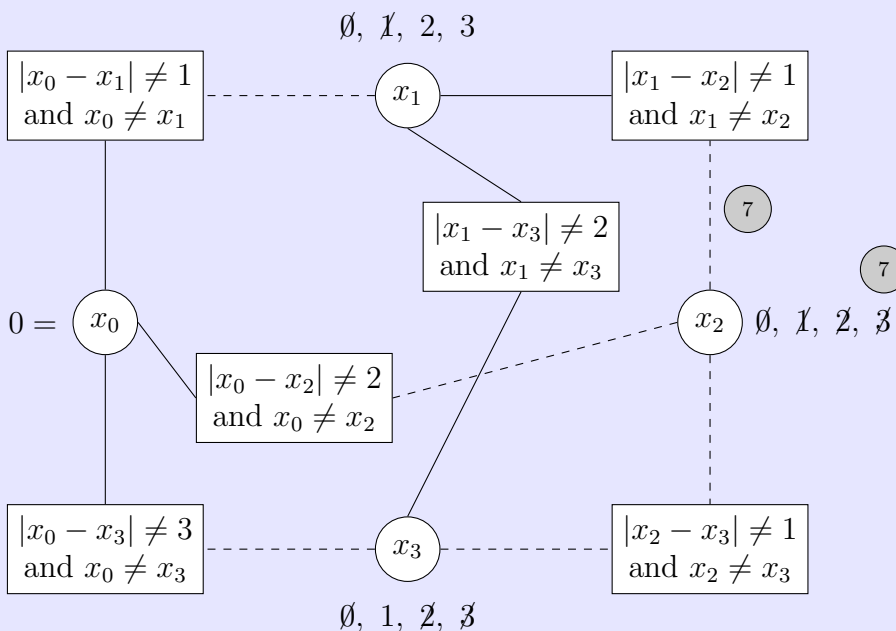
The domains of the variables become:  $x_0 \in \{0\}$ ,  $x_1 \in \{2, 3\}$ ,  $x_2 \in \{3\}$ ,  $x_3 \in \{1\}$ .



**Step 7: Remove  $\langle x_2, c(x_1, x_2) \rangle$  from the set  $S$ .**

$\langle x_2, c(x_1, x_2) \rangle$  is NOT arc-consistent. We need to remove 3 from the domain of  $x_2$ .

The domain of  $x_2$  is empty. The AC-3 algorithm terminates and there is no solution (if we start with the initial assignment of  $x_0 = 0$ ).



## 6.3 Properties of the AC-3 algorithm

**Does the order in which the arcs are considered matter?**

No. Any order will lead to the same solution.

**Three possible outcomes of the AC-3 algorithm**

1. A domain is empty. No solution exists.
2. Every domain has one value left. A unique solution exists.
3. Every domain has at least one value left and at least one domain has multiple values left. AC-3 inconclusive; need to do search or split domains to determine whether a solution exists.

**Is AC-3 guaranteed to terminate?**

Yes.

**What is the complexity of the AC-3 algorithm?**

Assume that the CSP has  $n$  variables, each variable's domain has at most  $d$  values, and there are  $c$  binary constraints.

At the beginning, there are  $2c$  arcs in the set. For each arc, checking whether the arc is consistent requires  $\mathcal{O}(d^2)$  time. (For each value in one domain, we need to look through each value in the other domain.)

After making an arc consistent and removing it from the set, we may need to add it back. Each arc may be added to the set  $d$  times. (To remove an arc from the set, we need to remove one value from a variable's domain. Since a domain has at most  $d$  values, we can only add an arc to the set at most  $d$  times.)

The overall run-time is  $\mathcal{O}(c \cdot d^2 \cdot d) = \mathcal{O}(cd^3)$ .

## 7 Practice Problems

1. Consider the constraint “ $X$  is divisible by  $Y$ ” between two variables  $X$  and  $Y$ . For each scenario below, determine whether the arc  $\langle X, c(X, Y) \rangle$  is consistent or not.
  - (1)  $D_X = \{10, 12\}$ ,  $D_Y = \{3, 5\}$
  - (2)  $D_X = \{10, 12\}$ ,  $D_Y = \{2\}$
  - (3)  $D_X = \{10, 12\}$ ,  $D_Y = \{3\}$
  - (4)  $D_X = \{10, 12\}$ ,  $D_Y = \{3, 5, 8\}$
2. (True or False) Let  $c$  be a binary constraint between  $X$  and  $Y$ . If  $\langle X, c(X, Y) \rangle$  is arc-consistent, then  $\langle Y, c(X, Y) \rangle$  is arc-consistent.
3. (True or False) Let  $c$  be a binary constraint between  $X$  and  $Y$ . If  $\langle X, c(X, Y) \rangle$  is arc-consistent, then, after removing a value from  $D_Y$ ,  $\langle X, c(X, Y) \rangle$  is still arc-consistent.
4. (True or False) Let  $c$  be a binary constraint between  $X$  and  $Y$ . If  $\langle X, c(X, Y) \rangle$  is arc-consistent, then, after removing a value from  $D_X$ ,  $\langle X, c(X, Y) \rangle$  is still arc-consistent.
5. Why don't we add the same constraint with the other variable as the primary variable back into the set  $S$ ? Prove that there is no need to add the other arc — if the arc was consistent before the change, it is still consistent.
6. For the third outcome of AC-3, come up with an example where every domain has at least one value left, at least one domain has multiple values left and the problem has no solutions.
7. Prove or disprove: If  $\langle X, c(X, Y) \rangle$  is arc-consistent, then  $\langle Y, c(X, Y) \rangle$  is arc-consistent.