Constraint Satisfaction

Chapter 6

6.4 Give precise formulations for each of the following as constraint satisfaction problems:

a. Rectilinear floor-planning: find non-overlapping places in a large rectangle for a number of smaller rectangles.

b. Class scheduling: There is a fixed number of professors and classrooms, a list of classes to be offered, and a list of possible time slots for classes. Each professor has a set of classes that he or she can teach.

c. Hamiltonian tour: given a network of cities connected by roads, choose an order to visit all cities in a country without repeating any.

a. For rectilinear floor-planning, one possibility is to have a variable for each of the small rectangles, with the value of each variable being a 4-tuple consisting of the x and y coordinates of the upper left and lower right corners of the place where the rectangle will be located. The domain of each variable is the set of 4-tuples that are the right size for the corresponding small rectangle and that fit within the large rectangle. Constraints say that no two rectangles can overlap; for example if the value of variable R1 is [0, 0, 5, 8], then no other variable can take on a value that overlaps with the 0, 0 to 5, 8 rectangle.

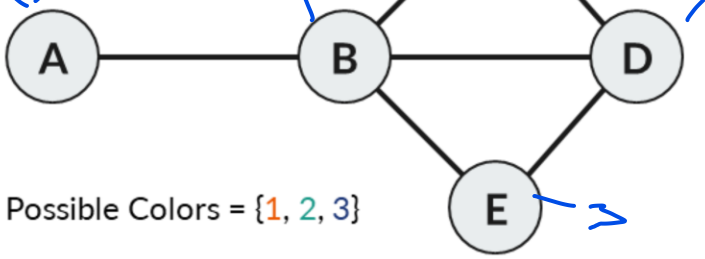
In more details:
The variable for each rectangle i is $V_i = (X_i, Y_i, W_i, H_i)$ where $0 \le X_i \le W$, $0 \le Y_i \le H$, $(Wi, Hi) \in \{ (S_{1i}, S_{2i}), (S_{2i}, S_{1i}) \}$ where W & H are the width & height of the large rectangle and S_{1i} & S_{2i} are the side lengths of rectangle i.
The constraints are that for each pair of rectangles i & j: $X_i + W_i \le X_j \vee Yi + H_i \le Y_j \vee X_j + W_j \le X_i \vee Y_j + H_j \le Y_i$

b. For class scheduling, one possibility is to have three variables for each class, one with times for values (e.g. MWF8:00, TuTh8:00, MWF9:00, ...), one with classrooms for values (e.g. Wheeler110, Evans330, ...) and one with instructors for values (e.g. Abelson, Bibel, Canny, ...). Constraints say that only one class can be in the same classroom at the same time, and an instructor can only teach one class at a time. There may be other constraints as well (e.g. an instructor should not have two consecutive classes).

In more details:
The variable for each class i can be $V_i = (T_i, C_i, I_i)$ where $T_i \in$ Possible Time Slots, $C_i \in$ Possible Classrooms and $I_i \in$ Possible Instructors.
The constraints are that for each pair of classes i & j: $(T_i = T_j) \Rightarrow (C_i \neq C_j \wedge I_i \neq I_j)$
This solution assumes that each class will have 1 Time slot only. If we want each class to have a different length, we can let T_i be the start time and change the constraint to include that the 2 classes can't have the same classroom or instructor if their time periods overlap.

c. For Hamiltonian tour, one possibility is to have one variable for each stop on the tour, with binary constraints requiring neighboring cities to be connected by roads, and an AllDiff constraint that all variables have a different value.

In more details:
If we have N cities, then we will use N variables Vi where $i \in [1,N]$ & Vi $\in$ cities and the value Vi is the city that we will visit at step i.
The constraints are all values assigned for $\{V_1, V_2, .. V_N\}$ are different (not city is repeated) and that for every consecutive pair (V_i, V_{i+1}) in Roads (there must be a road connecting them).



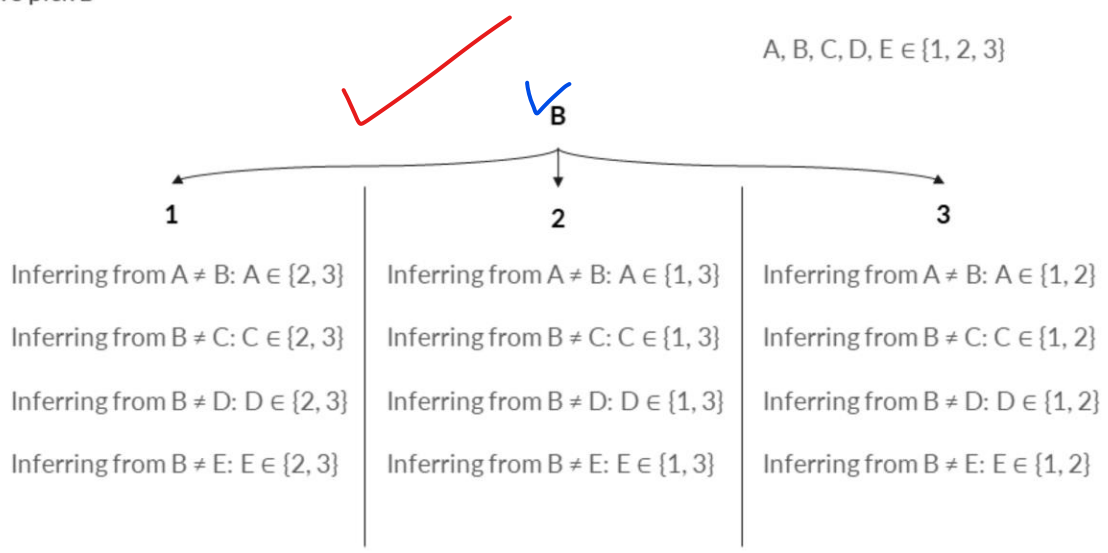
Possible Colors = {1, 2, 3}

Solve this Graph Coloring Problem by hand using the strategy of backtracking with forward checking and the degree & the least-constraining-value heuristics.

First, Define the domains and the constraints

Domains	Constraints
$A, B, C, D, E \in \{1, 2, 3\}$	$A \neq B, B \neq C, B \neq D,$
	$B \neq E, C \neq D, D \neq E$

Based on the degree heuristics, we pick the Variable with the greatest number of constraint.
So we pick B

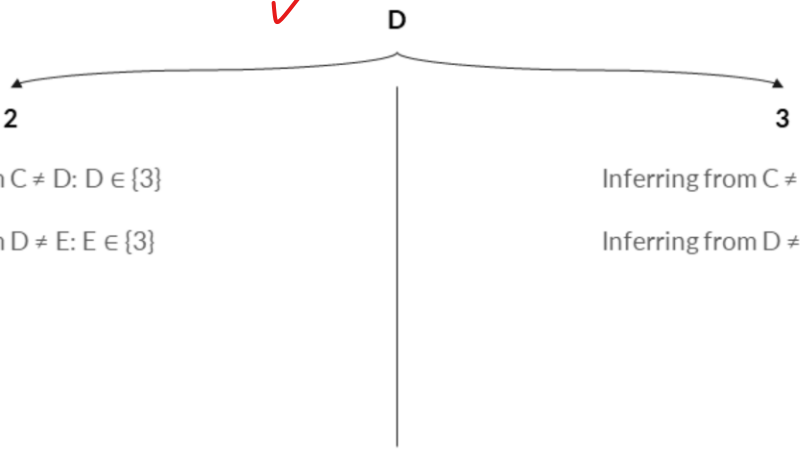


Based on the least constraining value, we choose the value that leaves the most options in the domains of other variables. But it doesn't matter here (all options leave 2*4 options in the other variables), so we choose 1 arbitrarily

Based on the degree heuristics, we pick D

Assignment:
B=1

A, C, D, E ∈ {2, 3}



Inferring from C ≠ D: D ∈ {3}

Inferring from D ≠ E: E ∈ {3}

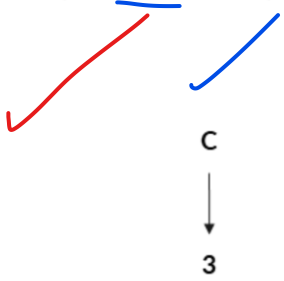
Inferring from C ≠ D: D ∈ {2}

Inferring from D ≠ E: E ∈ {2}

The choice doesn't matter here too, so we choose 2 arbitrarily.

Based on the degree heuristics, we either pick C or E. So we pick C arbitrarily.

Assignment:
B=1, D=2



$A \in \{2, 3\}$
$C, E \in \{3\}$

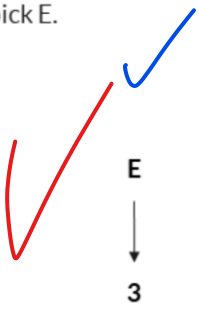
C

3

We only have 1 choice so we pick it.

Based on the degree heuristics, we either pick E.

Assignment:
B=1, D=2, C=3



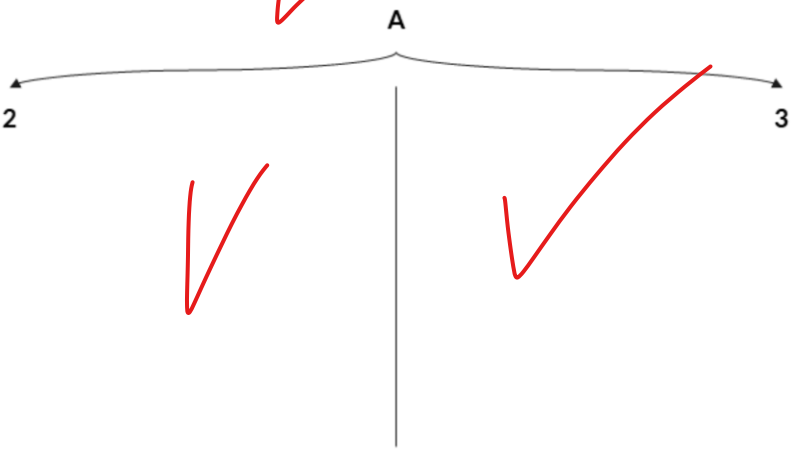
A ∈ {2, 3}
E ∈ {3}

We only have 1 choice so we pick it.

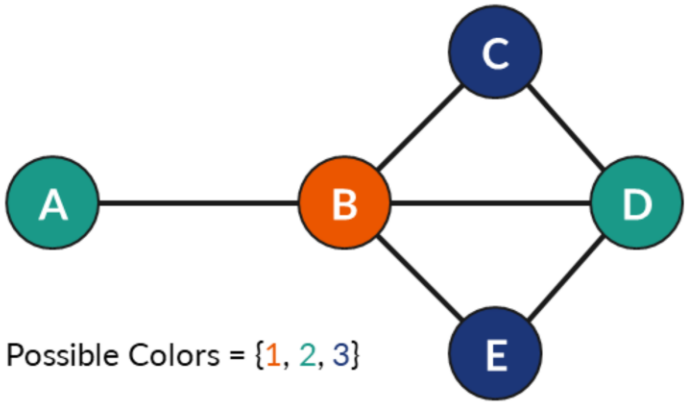
Based on the degree heuristics, we pick D

Assignment:
B=1, D=2, C=3, E=3

$A \in \{2, 3\}$



The choice doesn't matter here too, so we choose 2 arbitrarily.



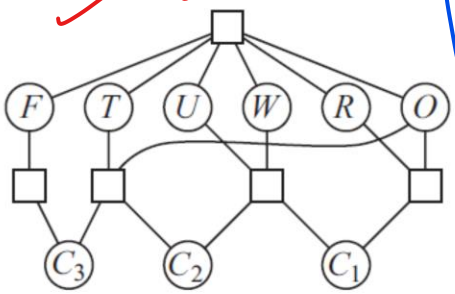
Possible Colors = {1, 2, 3}

Solution: B=1, D=2, C=3, E=3, A=2

No Backtracking was needed.

$$\begin{array}{r} T\ W\ O \\ +\ T\ W\ O \\ \hline F\ O\ U\ R \end{array}$$

6.5 Solve the cryptarithmetic problem in Figure 6.2 by hand, using the strategy of backtracking with forward checking and the MRV and least-constraining-value heuristics.



Solution in the upcoming slides

First, Define the domains and the constraints

Domains

C1, C2, C3 ∈ {0, 1}

F, T ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9}

W, O, U, R ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

Constraints

All-Different {F, T, W, O U, R}

F = C3

10*C3 + O = 2*T + C2

10*C2 + U = 2*W + C1

10*C1 + R = 2*O

Based on MRV heuristics, we pick the Variable with the least remaining values in its domain.
So we either pick C1, C2 or C3 since all of them have only 2 possible values.
Let's pick C3 arbitrarily.

$C1, C2, C3 \in \{0, 1\}$
$F, T \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
$W, O, U, R \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

C3

0

Inferring from F = C3:

$F \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\} \cap \{0\} \in \varphi$

Stop. It is now impossible to assign a value for F so no need to continue.

1

Inferring from F = C3:

$F \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\} \cap \{1\} \in \{1\}$

Inferring from 10*C3 + O = 2*T + C2:

$T \in \{5, 6, 7, 8, 9\}$

Based on the least constraining value, we choose the value that leaves the most options in the domains of other variables. We will use Forward Checking for that. We choose **"1"** because 0 leads to a contradiction.

Now, F has the least remaining values, so we pick it. No need for the least constraining value heuristic but we are doing Forward Checking anyway.

C3 = 1

F

↓

1

C1, C2 ∈ {0, 1}
F ∈ {1}
T ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9}
W, O, U, R ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

Inferring from All-Different{F, T, W, O U, R}:

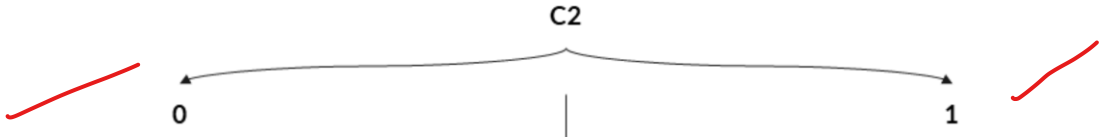
T ∈ {5, 6, 7, 8, 9}

W, O, U, R ∈ {0, 2, 3, 4, 5, 6, 7, 8, 9}

Now we either pick C1 or C2 since both of them have only 2 possible values.
Let's pick C2 arbitrarily.

C3 = 1, F = 1

C1, C2 ∈ {0, 1}
T ∈ {5, 6, 7, 8, 9}
W, O, U, R ∈ {0, 2, 3, 4, 5, 6, 7, 8, 9}



Inferring from 10*1 + O = 2*T + C2:

O ∈ {0, 2, 4, 6, 8}, T ∈ {5, 6, 7, 8, 9}

Inferring from 10*C2 + U = 2*W + C1

W ∈ {0, 2, 3, 4}

Inferring from 10*1 + O = 2*T + C2:

O ∈ {3, 5, 7, 9}, T ∈ {6, 7, 8, 9}

Inferring from 10*C2 + U = 2*W + C1

W ∈ {5, 6, 7, 8, 9}

Based on the least constraining value, we choose **"0"** because it leaves us with 14 possible values for its neighbors while 1 leaves use with 13.

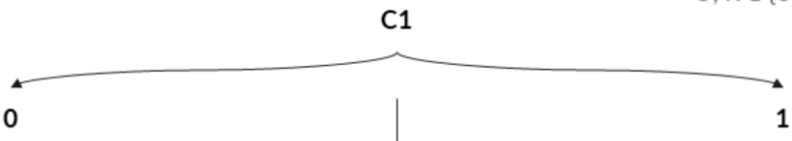
Revised

11/1/2024

Now we pick C1 since it have only 2 possible values.

C3 = 1, F = 1, C2 = {0, 1}

C1 ∈ {0, 1}
T ∈ {5, 6, 7, 8, 9}
O ∈ {0, 2, 4, 6, 8}
W ∈ {0, 2, 3, 4}
U, R ∈ {0, 2, 3, 4, 5, 6, 7, 8, 9}



Inferring from 10*0 + U = 2*W + C1:

U ∈ {0, 4, 6, 8}, W ∈ {0, 2, 3, 4}

Inferring from 10*C1 + R = 2*O:

O ∈ {0, 2, 4}, R ∈ {0, 4, 8}

Inferring from 10*0 + U = 2*W + C1:

U ∈ {5, 7, 9}, W ∈ {2, 3, 4}

Inferring from 10*C1 + R = 2*O:

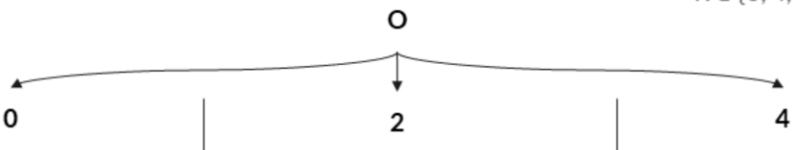
O ∈ {6, 8}, R ∈ {2, 6}

Based on the least constraining value, we choose **"0"** because it leaves us with 14 possible values for its neighbors while 1 leaves use with 10.

Now we either pick O or R since both of them have only 3 possible values. Let's pick O arbitrarily.

C3 = 1, F = 1, C2 = {0, 1}, C1 = {0, 1}

T ∈ {5, 6, 7, 8, 9}
O ∈ {0, 2, 4}
W ∈ {0, 2, 3, 4}
U ∈ {0, 4, 6, 8}
R ∈ {0, 4, 8}



Inferring from 10*0 + R = 2*O:
R ∈ {0}

Inferring from All-Different {F, T, W, O U, R}:
R ∈ φ **STOP**

Inferring from 10*0 + R = 2*O:
R ∈ {4}

Inferring from 10*1 + O = 2*T + 0:
T ∈ {6}

Inferring from All-Different {F, T, W, O U, R}:
T ∈ {6}, W ∈ {0, 3, 4}, U ∈ {0, 4, 6, 8}, R ∈ {4}

Inferring from 10*0 + R = 2*O:
R ∈ {8}

Inferring from 10*1 + O = 2*T + 0:
T ∈ {7}

Inferring from All-Different {F, T, W, O U, R}:
T ∈ {7}, W ∈ {0, 2, 3}, U ∈ {0, 6, 8}, R ∈ {8}

Based on the least constraining value, we choose **"2"**.

Now we either pick T or R since both of them have only 1 possible values. Let's pick T arbitrarily.

C3 = 1, F = 1, C2 = {0, 1}, C1 = {0, 1}, O = {2, 4}

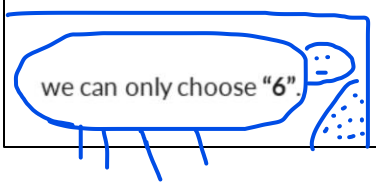
T ∈ {6}
W ∈ {0, 3, 4}
U ∈ {0, 4, 6, 8}
R ∈ {4}

T

↓

6

Inferring from All-Different {F, T, W, O U, R}:
W ∈ {0, 3, 4}, U ∈ {0, 4, 8}, R ∈ {4}



Now we pick R.

W ∈ {0, 3, 4}
U ∈ {0, 4, 8}
R ∈ {4}

C3 = 1, F = 1, C2 = {0, 1}, C1 = {0, 1}, O = {2, 4}, T = 6

R

↓

4

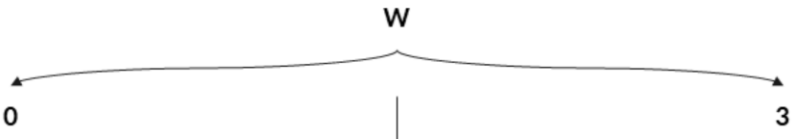
Inferring from All-Different {F, T, W, O U, R}:
W ∈ {0, 3}, U ∈ {0, 8}

we can only choose **"4"**.

Now we either pick W or U since both of them have only 2 possible values. Let's pick W arbitrarily.

$W \in \{0, 3\}$
$U \in \{0, 8\}$

C3 = 1, F = 1, C2 = {0, 1}, C1 = {0, 1}, O = {2, 4}, T = 6, R = 4



Inferring from 10*0 + U = 2*W + 0:

$U \in \{0\}$

Inferring from All-Different{F, T, W, O U, R}:

$U \in \varphi$ **STOP**

Inferring from 10*0 + U = 2*W + 0:

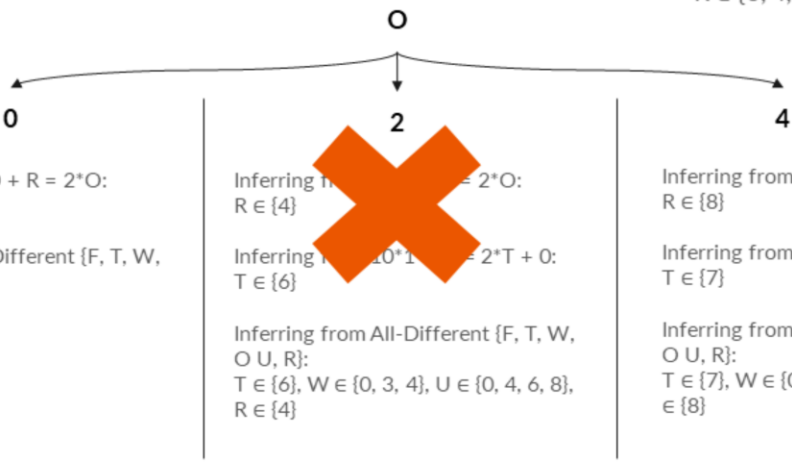
$U \in \{0, 8\} \cap \{6\} \in \varphi$ **STOP**

We reached a dead end. So we back track to the latest variable with branches (which is O) and pick another branch.

Now we either pick O or R since both of them have only 3 possible values. Let's pick O arbitrarily.

C3 = 1, F = 1, C2 = {0, 1}, C1 = {0, 1}

T ∈ {5, 6, 7, 8, 9}
O ∈ {0, 2, 4}
W ∈ {0, 2, 3, 4}
U ∈ {0, 4, 6, 8}
R ∈ {0, 4, 8}



"2" lead to a dead end. So we choose **"4"**.

Now we either pick T or R since both of them have only 1 possible values. Let's pick T arbitrarily.

C3 = 1, F = 1, C2 = {0, 1}, C1 = {0, 1}, O = 4

T ∈ {7}
W ∈ {0, 2, 3}
U ∈ {0, 6, 8}
R ∈ {8}

T

↓

7

Inferring from All-Different {F, T, W, O U, R}:
W ∈ {0, 2, 3}, U ∈ {0, 6, 8}, R ∈ {8}

we can only choose **"7"**.

Now we pick R.

W ∈ {0, 2, 3}
U ∈ {0, 6, 8}
R ∈ {8}

C3 = 1, F = 1, C2 = {0, 1}, C1 = {0, 1}, O = 4, T = 7

R

↓

8

Inferring from All-Different {F, T, W, O U, R}:
W ∈ {0, 2, 3}, U ∈ {0, 6}

we can only choose **"8"**.

Now we pick U since both of them have only 2 possible values.

$W \in \{0, 2, 3\}$
$U \in \{0, 6\}$

C3 = 1, F = 1, C2 = {0, 1}, C1 = {0, 1}, O = O = {2, 4}, T = 7, R = 8

U

0

Inferring from 10*0 + U = 2*W + 0:

$W \in \{0\}$

Inferring from All-Different{F, T, W, O U, R}:

$W \in \varphi$ **STOP**

6

Inferring from 10*0 + U = 2*W + 0:

$W \in \{3\}$

Inferring from All-Different{F, T, W, O U, R}:

$W \in \{3\}$

We pick "6" since it is the only value that doesn't cause a contradiction.

Finally, we pick W.

$W \in \{3\}$

C3 = 1, F = 1, C2 = {0, 1}, C1 = {0, 1}, O = 4, T = 7, R = 8, U = 6

W

↓

3

We reached the end

we can only choose **"3"**.

C3 = 1, F = 1, C2 = 0, C1 = 0, O = 4, T = 7, R = 8, U = 6, W = 3

$$\begin{array}{r} T\ W\ O \\ +\ T\ W\ O \\ \hline F\ O\ U\ R \end{array}$$

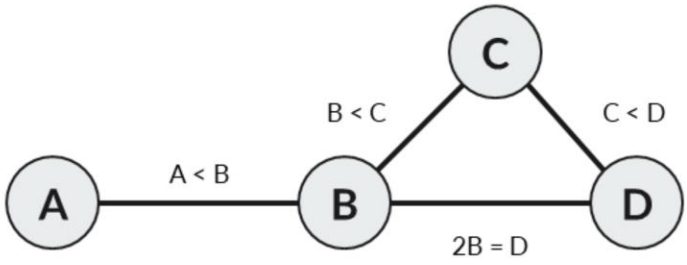
$$\begin{array}{r} 734 \\ +\ 734 \\ \hline 1468 \end{array}$$

6.6 Show how a single ternary constraint such as "$A + B = C$" can be turned into three binary constraints by using an auxiliary variable. You may assume finite domains. (*Hint:* Consider a new variable that takes on values that are pairs of other values, and consider constraints such as "X is the first element of the pair Y.") Next, show how constraints with more than three variables can be treated similarly. Finally, show how unary constraints can be eliminated by altering the domains of variables. This completes the demonstration that any CSP can be transformed into a CSP with only binary constraints.

The problem statement sets out the solution fairly completely. To express the ternary constraint on A, B and C that A + B = C, we first introduce a new variable, AB. If the domain of A and B is the set of numbers N, then the domain of AB is the set of pairs of numbers from N, i.e. N × N. Now there are three binary constraints, one between A and AB saying that the value of A must be equal to the first element of the pair-value of AB; one between B and AB saying that the value of B must equal the second element of the value of AB; and finally one that says that the sum of the pair of numbers that is the value of AB must equal the value of C. All other ternary constraints can be handled similarly.

Now that we can reduce a ternary constraint into binary constraints, we can reduce a 4-ary constraint on variables A,B,C,D by first reducing A,B,C to binary constraints as shown above, then adding back D in a ternary constraint with AB and C, and then reducing this ternary constraint to binary by introducing CD.

By induction, we can reduce any n-ary constraint to an (n − 1)-ary constraint. We can stop at binary, because any unary constraint can be dropped, simply by moving the effects of the constraint into the domain of the variable.



Possible Values = {1, 2, 3, 4, 5}

Apply AC-3 to this CSP.

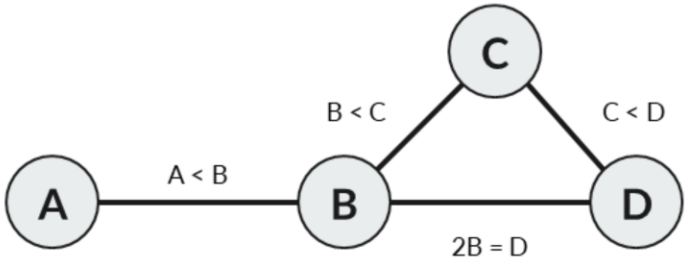
```
function AC-3( csp) returns false if an inconsistency is found and true otherwise
  inputs: csp, a binary CSP with components (X, D, C)
  local variables: queue, a queue of arcs, initially all the arcs in csp

  while queue is not empty do
    (X_i, X_j) ← REMOVE-FIRST(queue)
    if REVISE(csp, X_i, X_j) then
      if size of D_i = 0 then return false
      for each X_k in X_i.NEIGHBORS - {X_j} do
        add (X_k, X_i) to queue
  return true
```

```
function REVISE( csp, X_i, X_j) returns true iff we revise the domain of X_i
  revised ← false
  for each x in D_i do
    if no value y in D_j allows (x,y) to satisfy the constraint between X_i and X_j then
      delete x from D_i
      revised ← true
  return revised
```

Since it is not useful to have the same arc more than once in the queue, I will treat the queue as a set and will not add arcs that are already inside the queue.

Another way to write it: https://en.wikipedia.org/wiki/AC-3_algorithm

Domains: A, B, C, D ∈ {1, 2, 3, 4, 5}



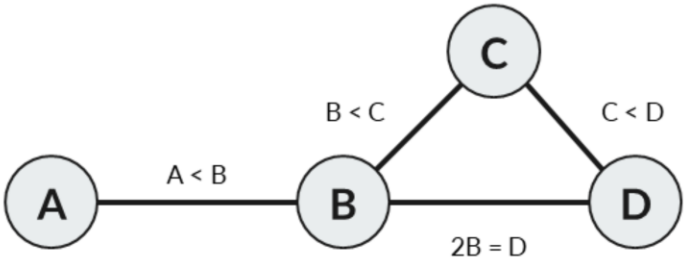
ToDo: {**(A, B)**, (B, A), (B, C), (C,B), (C, D), (D, C), (B, D), (D, B)}

For A ∈ {1, 2, 3, 4, 5}, B ∈ {1, 2, 3, 4, 5}

There is no value for B that satisfies A<B if A=5.

So A ∈ {1, 2, 3, 4} and there is nothing to add to ToDo.

Domains:
A ∈ {1, 2, 3, 4}
B, C, D ∈ {1, 2, 3, 4, 5}



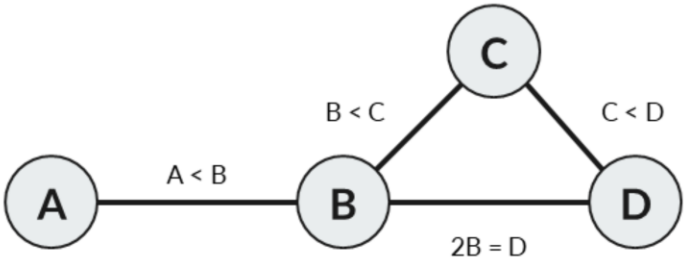
ToDo: {**(B, A)**, (B, C), (C,B), (C, D), (D, C), (B, D), (D, B)}

For B ∈ {1, 2, 3, 4, 5}, A ∈ {1, 2, 3, 4}

There is no value for A that satisfies A<B if B=1.

So B ∈ {2, 3, 4, 5} and add (C, B) & (D, B) to ToDo (nothing new will be added here).

Domains:
A ∈ {1, 2, 3, 4}
B ∈ {2, 3, 4, 5}
C, D ∈ {1, 2, 3, 4, 5}



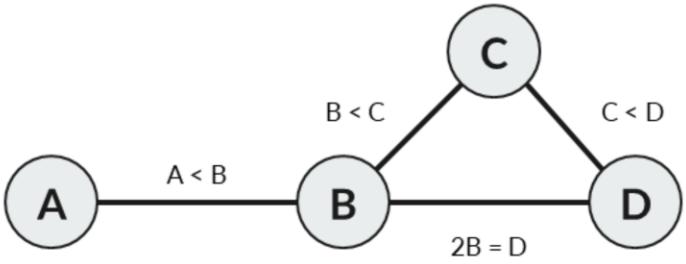
ToDo: {**(B, C)**, (C,B), (C, D), (D, C), (B, D), (D, B)}

For B ∈ {2, 3, 4, 5}, C ∈ {1, 2, 3, 4, 5}

There is no value for C that satisfies B<C if B=5.

So B ∈ {2, 3, 4} and add (A, B) & (D, B) to ToDo (only (A, B) will be added).

Domains:
A ∈ {1, 2, 3, 4}
B ∈ {2, 3, 4}
C, D ∈ {1, 2, 3, 4, 5}



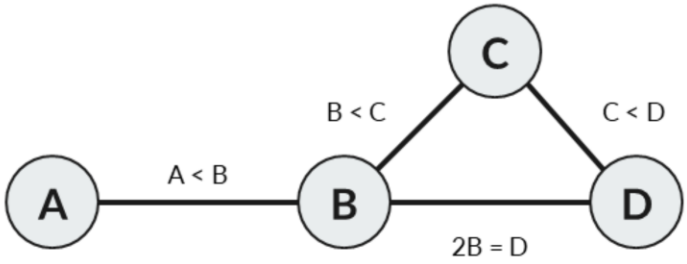
ToDo: {**(C,B)**, (C, D), (D, C), (B, D), (D, B), (A, B)}

For C ∈ {1, 2, 3, 4, 5}, B ∈ {2, 3, 4}

There is no value for B that satisfies B<C if C=1 or C=2.

So C ∈ {3, 4, 5} and add (D, C) to ToDo (nothing new will be added here).

Domains:
A ∈ {1, 2, 3, 4}
B ∈ {2, 3, 4}
C ∈ {3, 4, 5}
D ∈ {1, 2, 3, 4, 5}



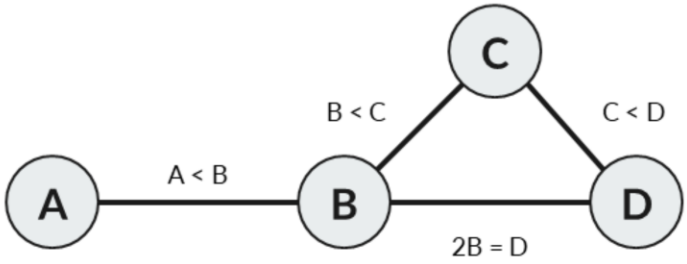
ToDo: {**(C, D)**, (D, C), (B, D), (D, B), (A, B)}

For C ∈ {3, 4, 5}, D ∈ {1, 2, 3, 4, 5}

There is no value for D that satisfies C<D if C=5.

So C ∈ {3, 4} and add (B, C) to ToDo.

Domains:
A ∈ {1, 2, 3, 4}
B ∈ {2, 3, 4}
C ∈ {3, 4}
D ∈ {1, 2, 3, 4, 5}



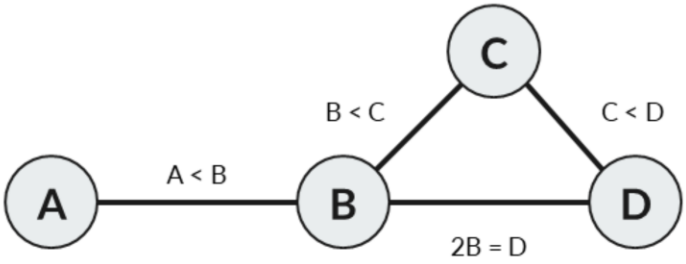
ToDo: {**(D, C)**, (B, D), (D, B), (A, B), (B, C)}

For D ∈ {1, 2, 3, 4, 5}, C ∈ {3, 4}

There is no value for C that satisfies C<D if D=1 or D=2 or D=3.

So D ∈ {4, 5} and add (B, D) to ToDo (nothing new will be added here).

Domains:
A ∈ {1, 2, 3, 4}
B ∈ {2, 3, 4}
C ∈ {3, 4}
D ∈ {4, 5}



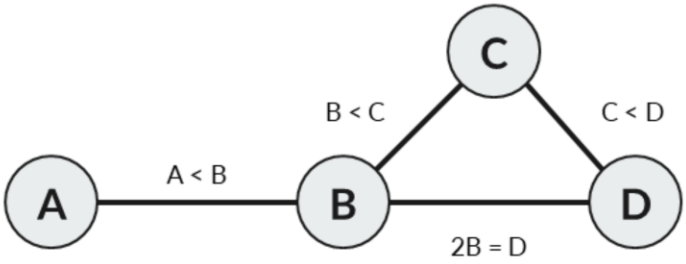
ToDo: {**(B, D)**, (D, B), (A, B), (B, C)}

For B ∈ {2, 3, 4}, D ∈ {4, 5},

There is no value for D that satisfies 2B=D if B=3 or B=4.

So B ∈ {2} and add (A, B) & (C, B) to ToDo (only (C, B) will be added here).

Domains:
A ∈ {1, 2, 3, 4}
B ∈ {2}
C ∈ {3, 4}
D ∈ {4, 5}



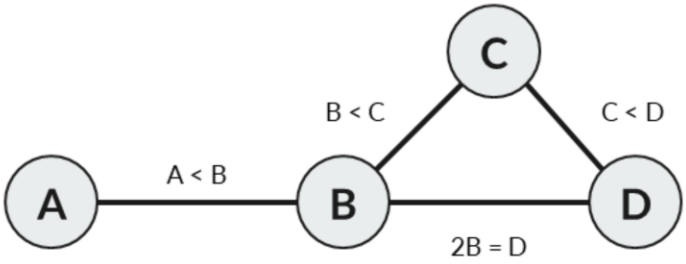
ToDo: {**(D, B)**, (A, B), (B, C), (C, B)}

For D ∈ {4, 5}, B ∈ {2},

There is no value for B that satisfies 2B=D if D=5.

So D ∈ {4} and add (C, D) to ToDo.

Domains:
A ∈ {1, 2, 3, 4}
B ∈ {2}
C ∈ {3, 4}
D ∈ {4}



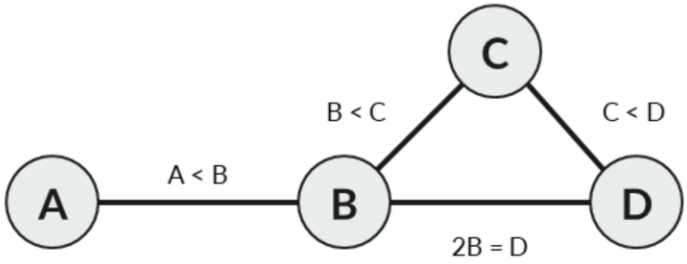
ToDo: {**(A, B)**, (B, C), (C, B), (C, D)}

For A ∈ {1, 2, 3, 4}, B ∈ {2},

There is no value for B that satisfies A < B if A=2 or A=3 or A=4.

So A ∈ {1} and add nothing to ToDo.

Domains:
A ∈ {1}
B ∈ {2}
C ∈ {3, 4}
D ∈ {4}

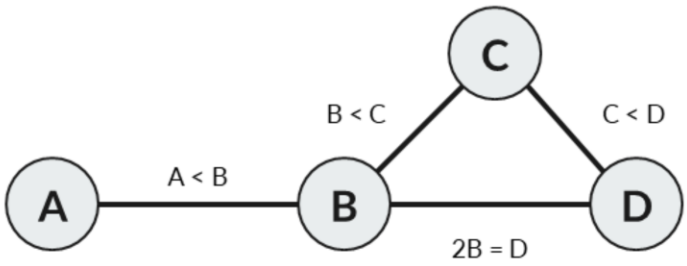


ToDo: {**(B, C)**, (C, B), (C, D)}

For B ∈ {2}, C ∈ {3, 4},

All combinations satisfy the constraints so skip.

Domains:
A ∈ {1}
B ∈ {2}
C ∈ {3, 4}
D ∈ {4}

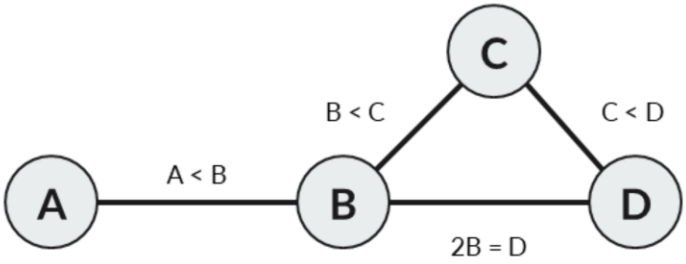


ToDo: {**(C, B)**, (C, D)}

For C ∈ {3, 4}, B ∈ {2},

All combinations satisfy the constraints so skip.

Domains:
A ∈ {1}
B ∈ {2}
C ∈ {3, 4}
D ∈ {4}



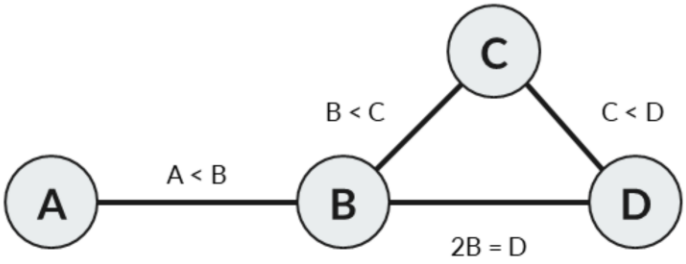
ToDo: {**(C, D)**}

For C ∈ {3, 4}, D ∈ {4},

There is no value for D that satisfies C < D if C=4.

So C ∈ {3} and add (B, C) to ToDo.

Domains:
A ∈ {1}
B ∈ {2}
C ∈ {3}
D ∈ {4}

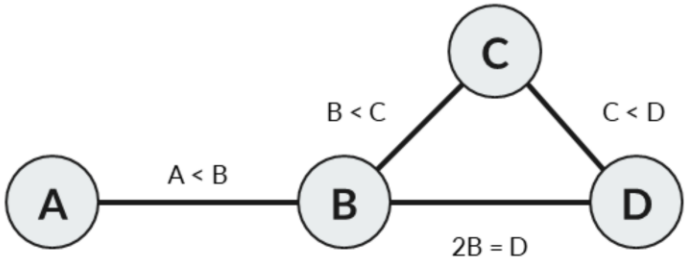


ToDo: {**(B, C)**}

For B ∈ {2}, C ∈ {3},

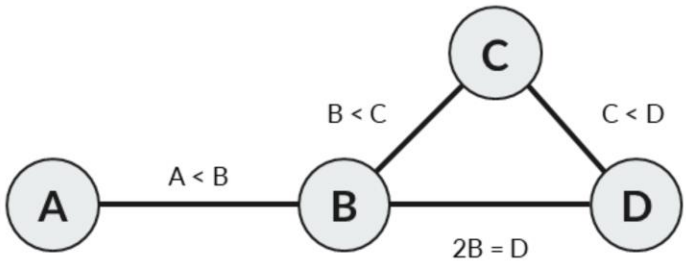
All combinations satisfy the constraints so skip.

Domains:
A ∈ {1}
B ∈ {2}
C ∈ {3}
D ∈ {4}



ToDo: {}

ToDo is empty so abort the algorithm.



Domains: $A \in \{1\}, B \in \{2\}, C \in \{3\}, D \in \{4\}$

Fortunately, AC-3 left us with only a single value in each domain so we don't even need to apply backtracking to find a solution for this problem.