Discrete Optimization

MA2827

Fondements de l'optimisation discrète

Constraint programming

https://project.inria.fr/2015ma2827/

Material based on the lectures of Pascal Van Hentenryck at Coursera

Outline

- Computational paradigm
- More constraints
  - linear constraints over integers
  - redundant constraints
  - symmetry breaking
- Global constraints
  - Feasibility
  - Pruning
- Search strategies

Constraint programming

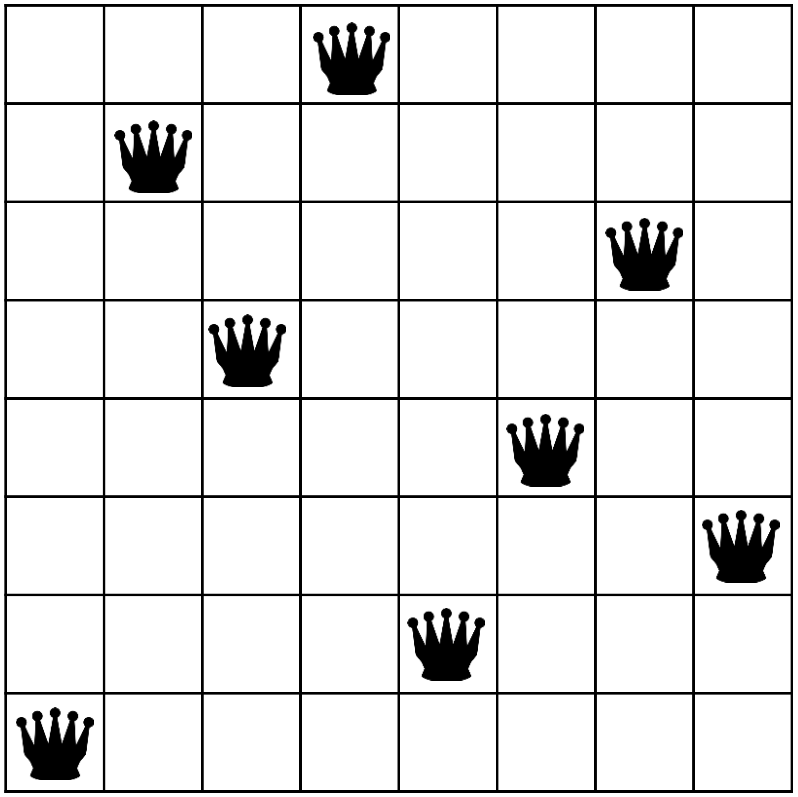
- Computational paradigm
  - use constraints to reduce the set of values that each variable can take
  - make a choice if no deduction can be made

- Modelling technology
  - convey the structure of the problem as explicitly as possible
  - express substructures of the problem
  - give solvers as much information as possible

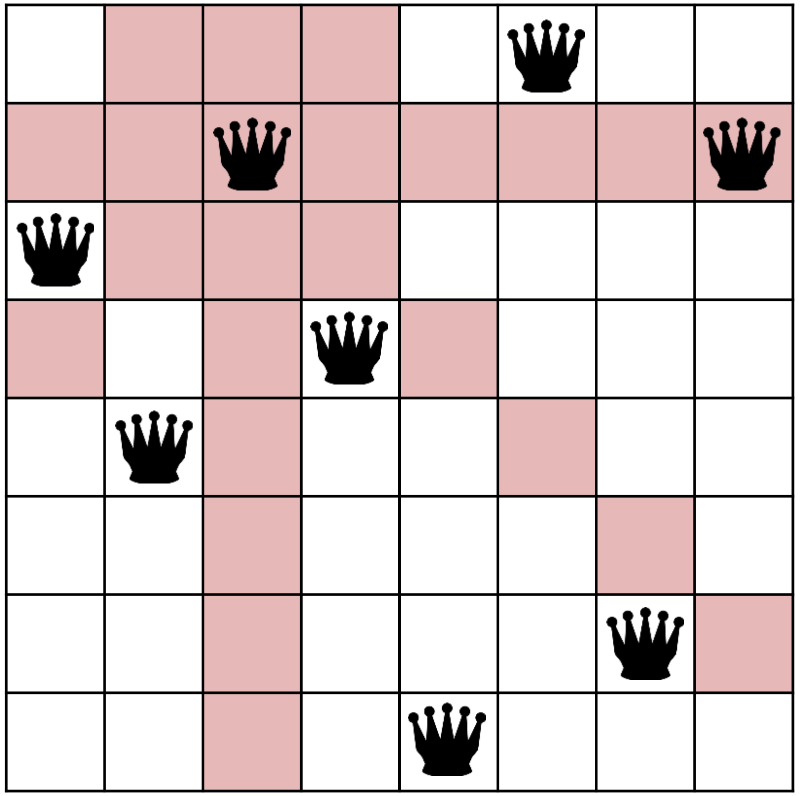
Example: 8-queen problem

Task: place 8 queens on the chess board such that they do not attack each other

Good



Bad

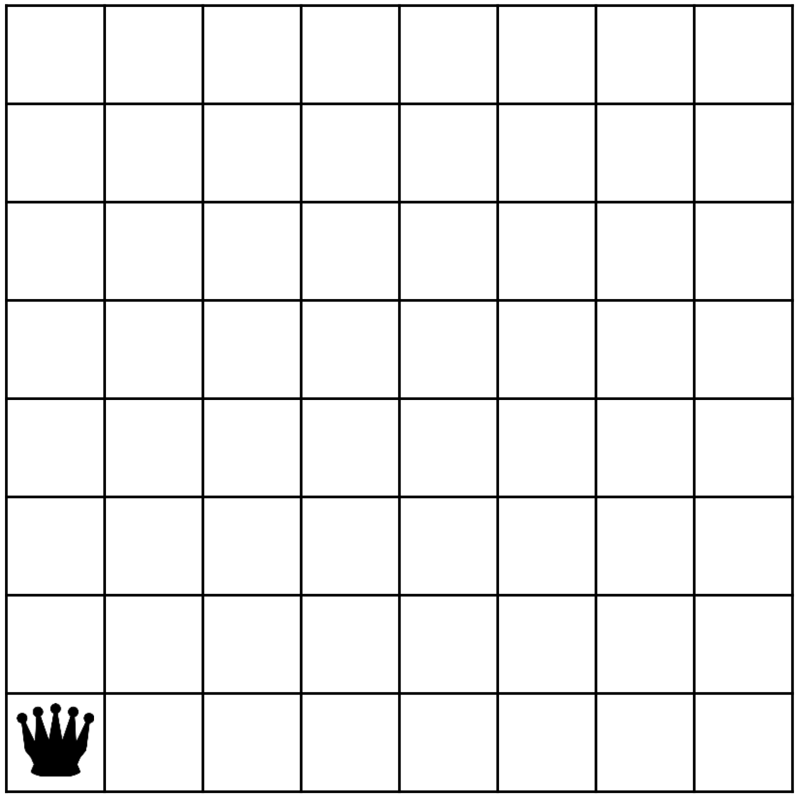


8 queens

Task: place 8 queens on the chess board such that they do not attack each other

Guess the first queen

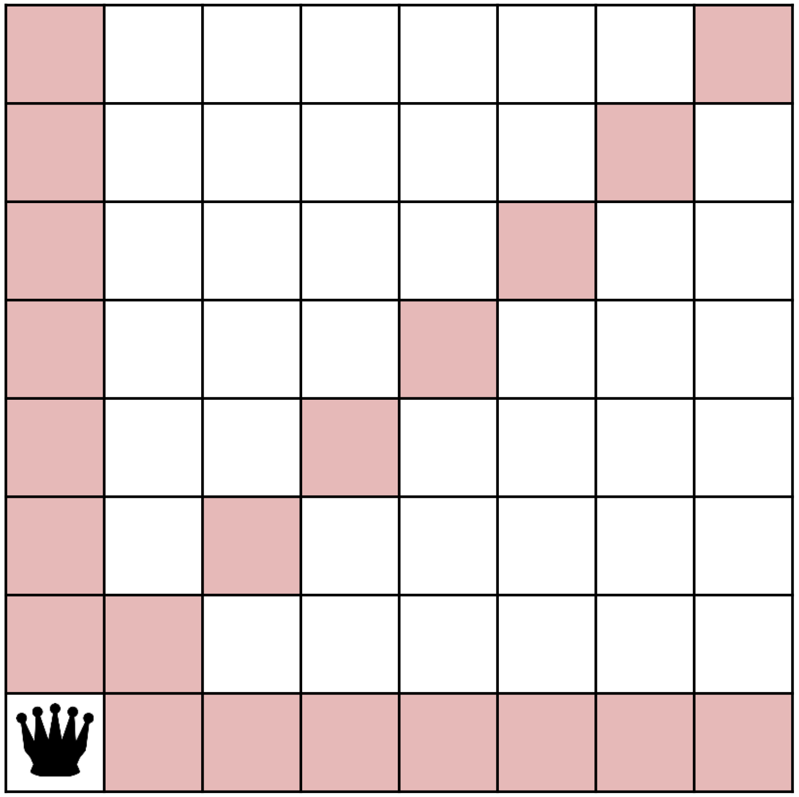
Try the first spot



8 queens

Task: place 8 queens on the chess board such that they do not attack each other

Guess the first queen



Apply constraints

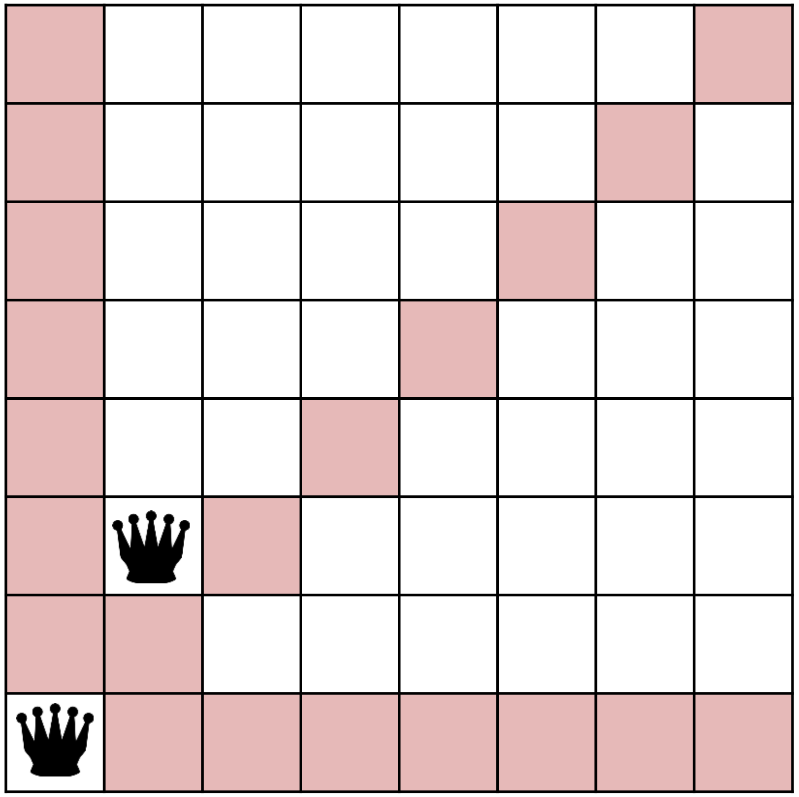
8 queens

Task: place 8 queens on the chess board such that they do not attack each other

Guess the first queen

Guess the second queen

Try the first available spot



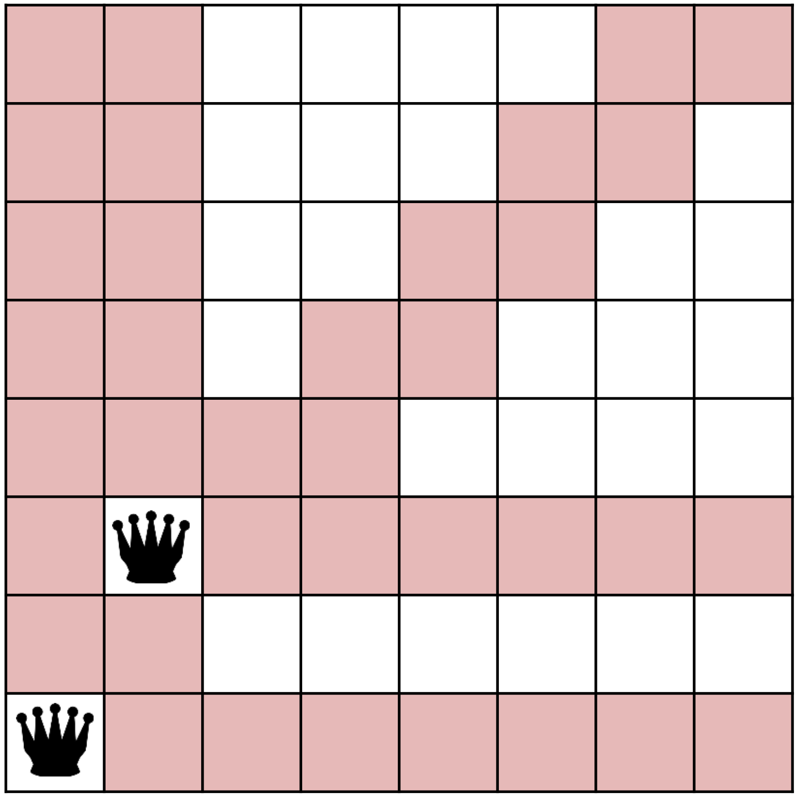
8 queens

Task: place 8 queens on the chess board such that they do not attack each other

Guess the first queen

Guess the second queen

Apply constraints



8 queens

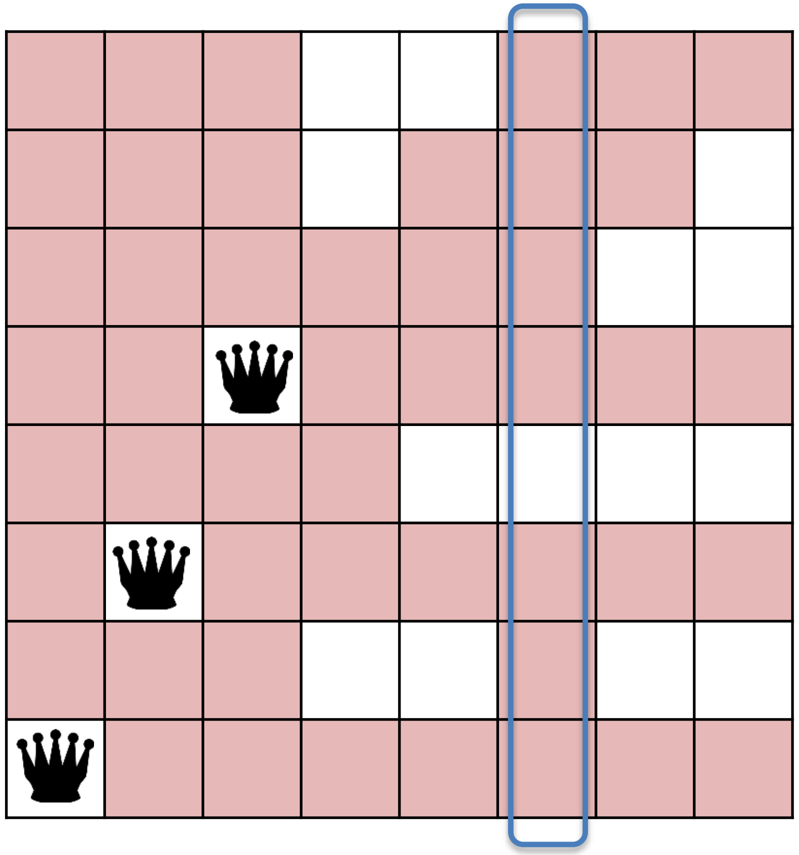
Task: place 8 queens on the chess board such that they do not attack each other

Guess the first queen

Guess the second queen

Guess the third queen

Apply constraints



8 queens

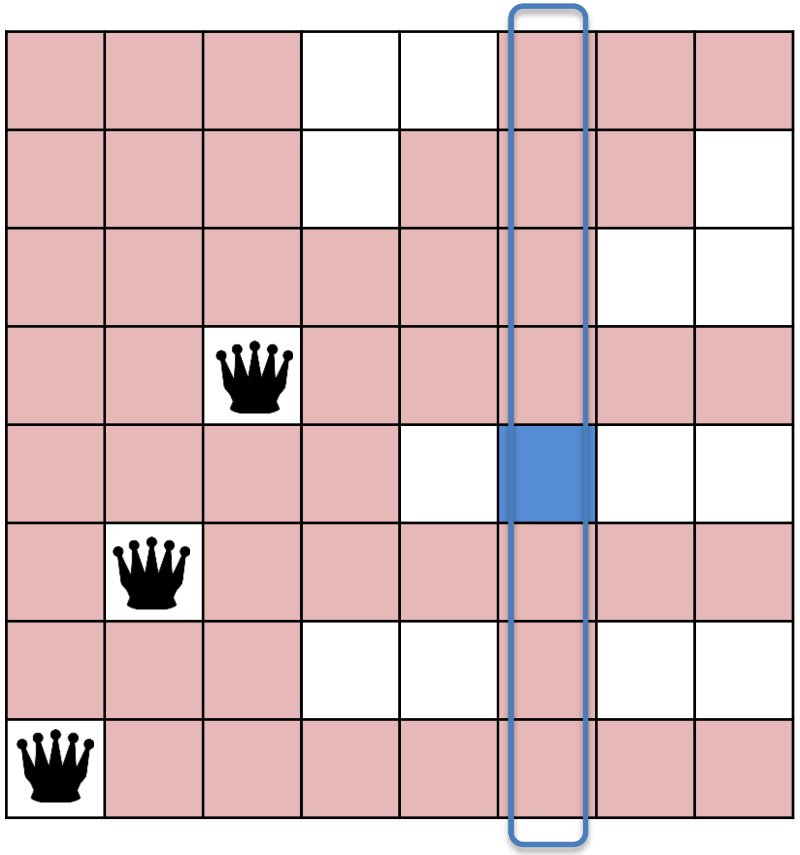
Task: place 8 queens on the chess board such that they do not attack each other

Guess the first queen

Guess the second queen

Guess the third queen

Only one possibility!



8 queens

Task: place 8 queens on the chess board such that they do not attack each other

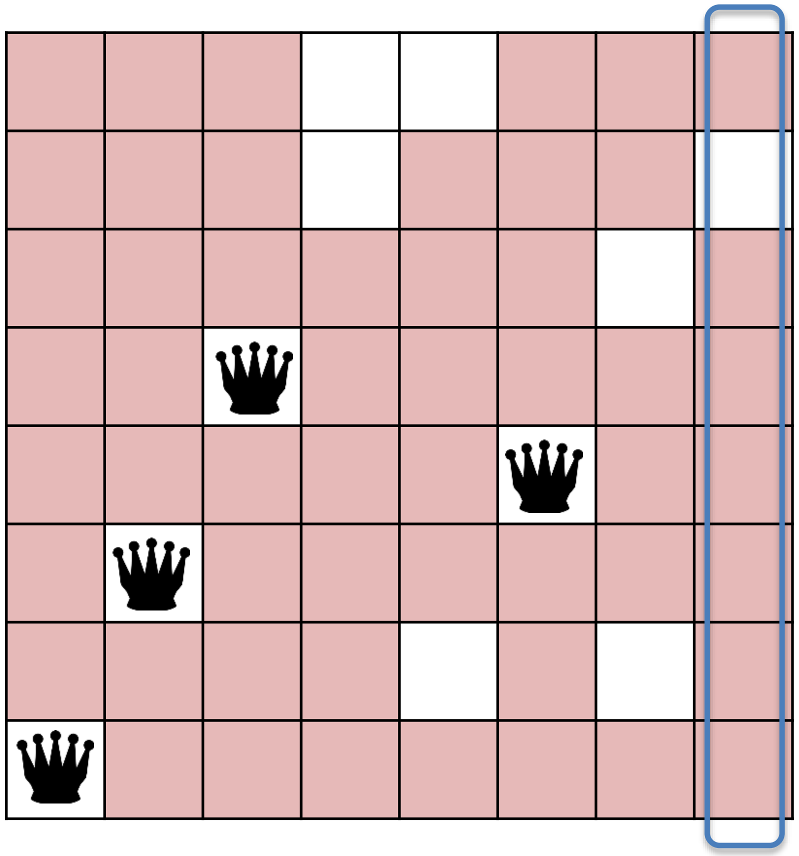
Guess the first queen

Guess the second queen

Guess the third queen

Place the fourth queen

Apply constraints



8 queens

Task: place 8 queens on the chess board such that they do not attack each other

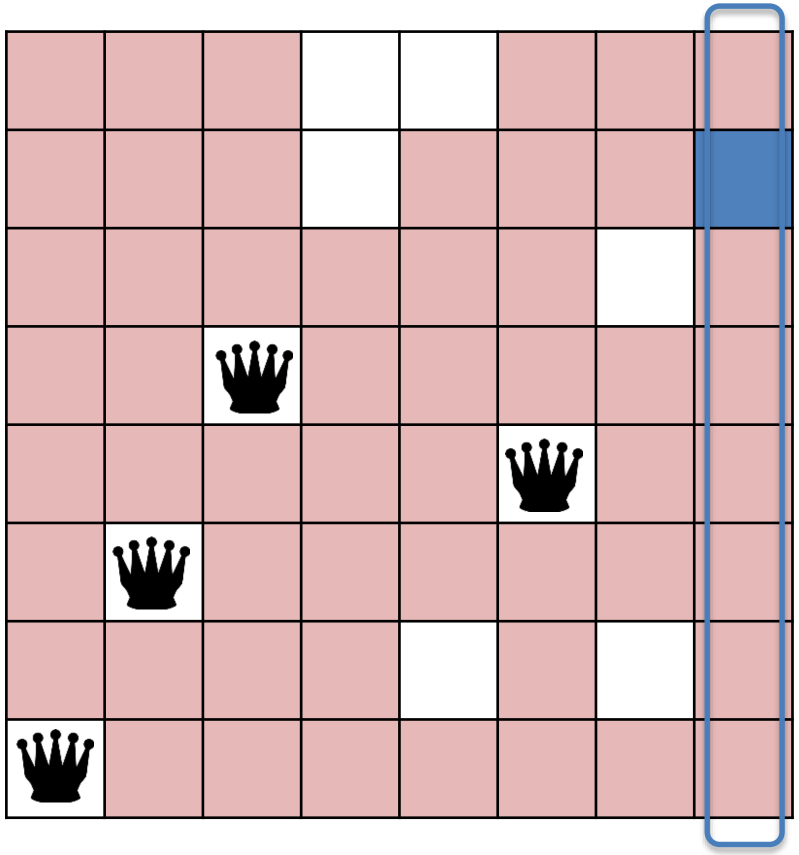
Guess the first queen

Guess the second queen

Guess the third queen

Place the fourth queen

Only one possibility!



8 queens

Task: place 8 queens on the chess board such that they do not attack each other

Guess the first queen

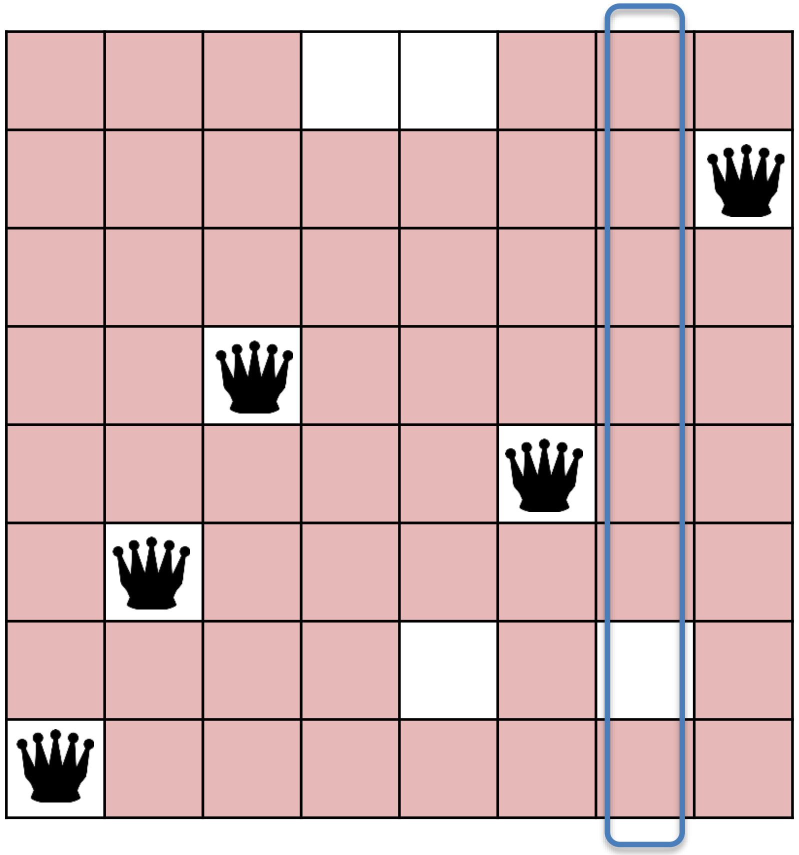
Guess the second queen

Guess the third queen

Place the fourth queen

Place the fifth queen

Apply constraints



8 queens

Task: place 8 queens on the chess board such that they do not attack each other

Guess the first queen

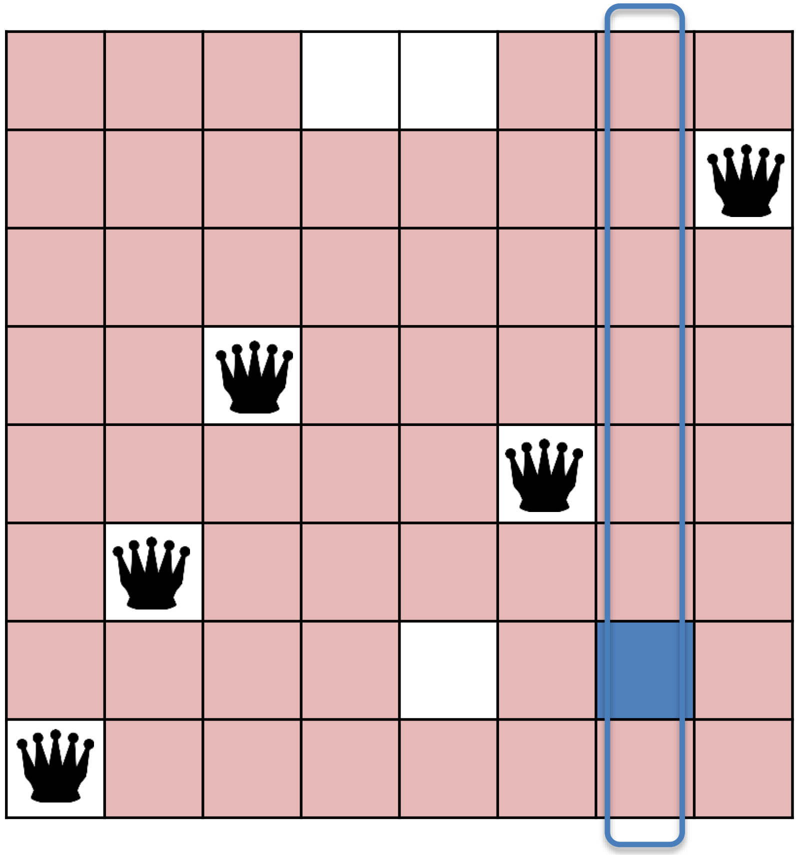
Guess the second queen

Guess the third queen

Place the fourth queen

Place the fifth queen

Only one possibility!



8 queens

Task: place 8 queens on the chess board such that they do not attack each other

Guess the first queen

Guess the second queen

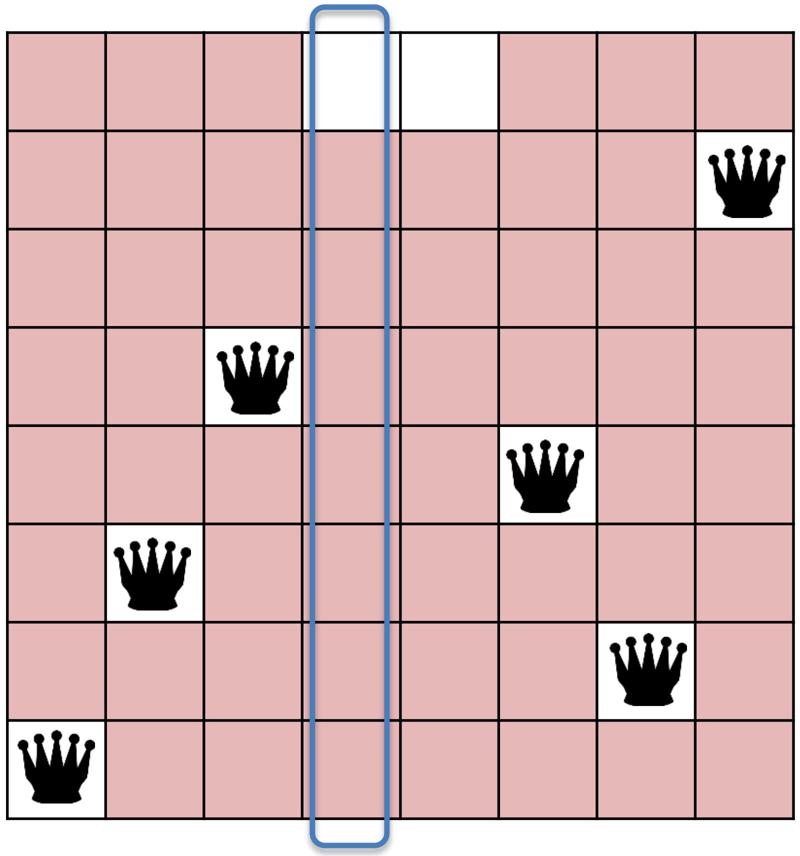
Guess the third queen

Place the fourth queen

Place the fifth queen

Place the sixth queen

Apply constraints



8 queens

Task: place 8 queens on the chess board such that they do not attack each other

Guess the first queen

Guess the second queen

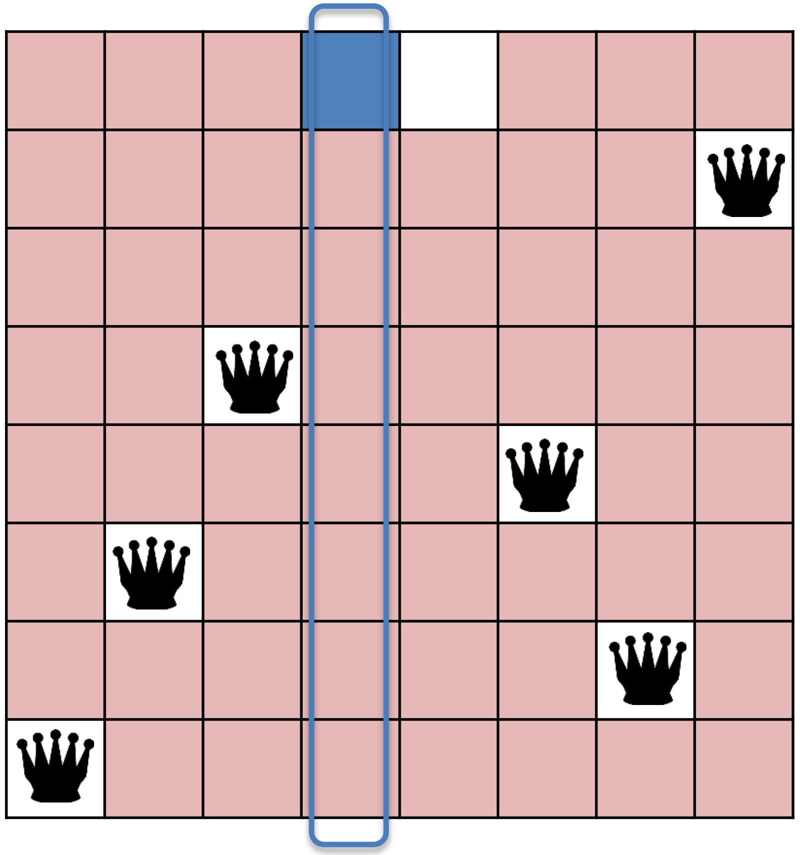
Guess the third queen

Place the fourth queen

Place the fifth queen

Place the sixth queen

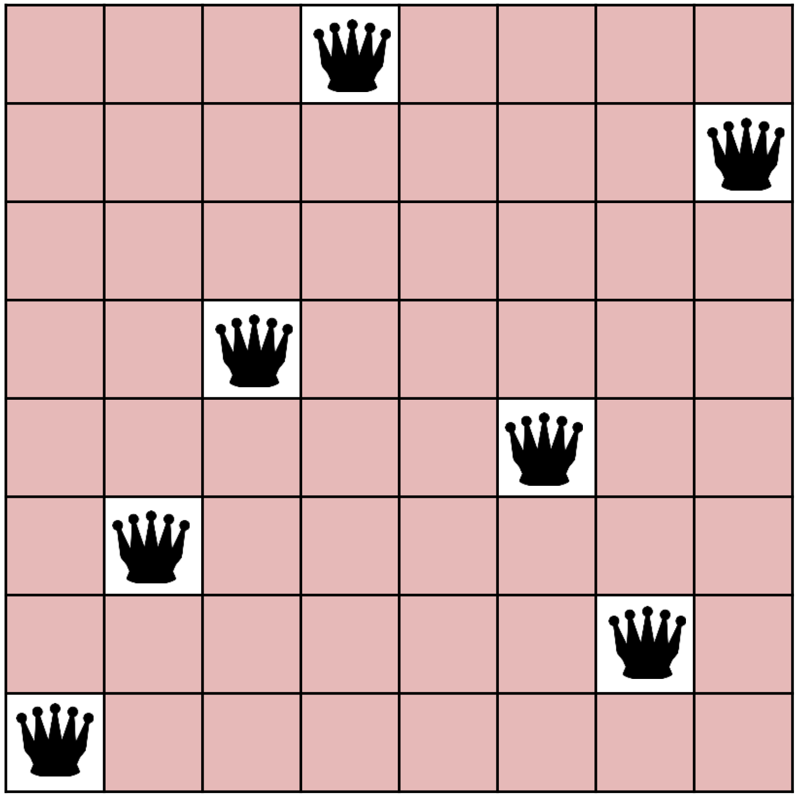
Only one possibility!



8 queens

Task: place 8 queens on the chess board such that they do not attack each other

Guess the first queen
Guess the second queen
Guess the third queen
Place the fourth queen
Place the fifth queen
Place the sixth queen
Place the seventh queen
Apply constraints



8 queens

Task: place 8 queens on the chess board such that they do not attack each other

Guess the first queen

Guess the second queen

Guess the third queen

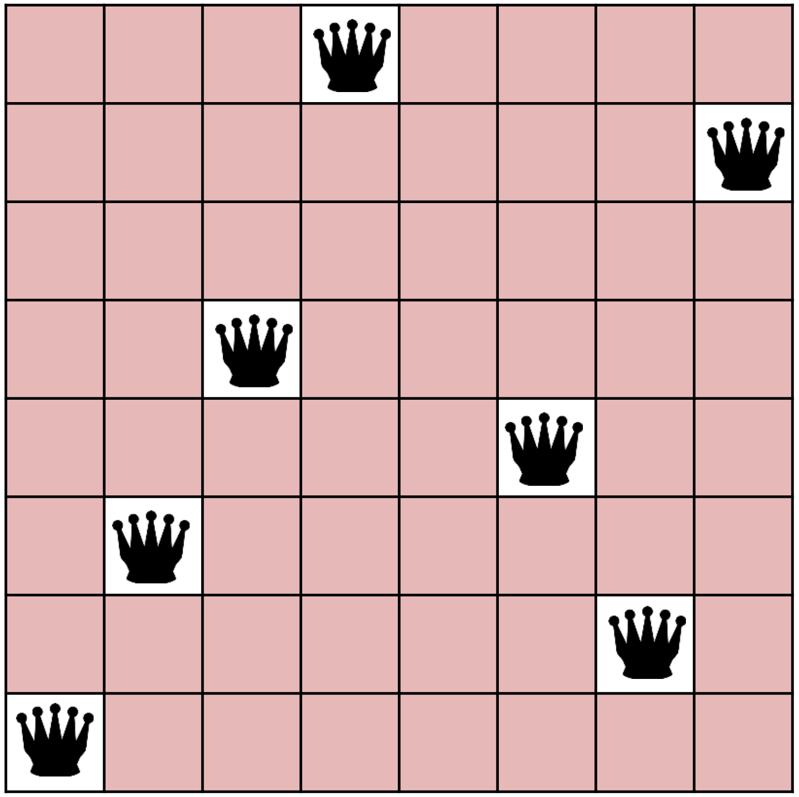
Place the fourth queen

Place the fifth queen

Place the sixth queen

Place the seventh queen

FAILURE!



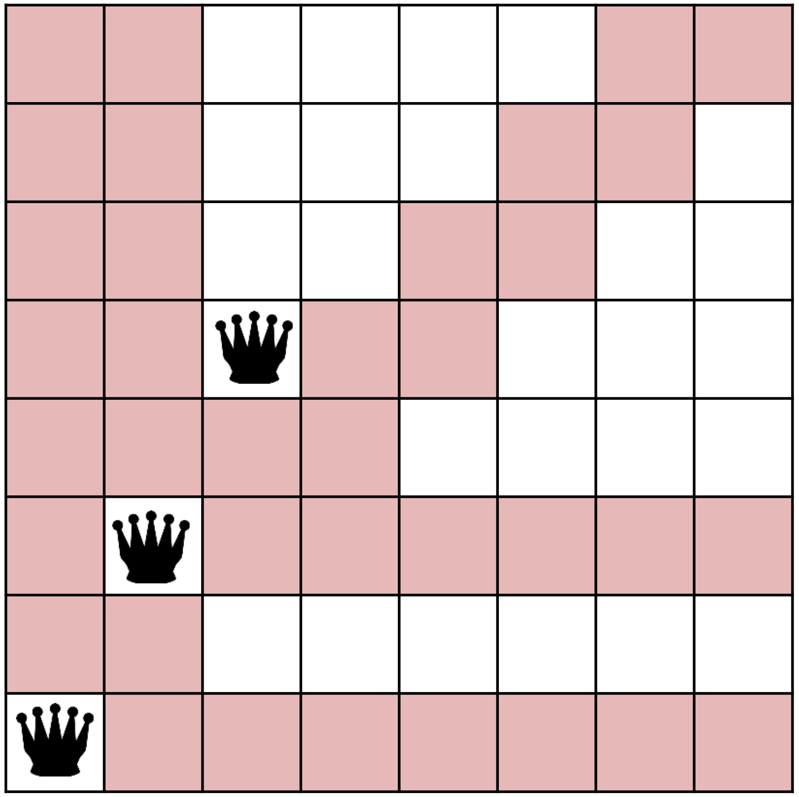
8 queens

Task: place 8 queens on the chess board such that they do not attack each other

Guess the first queen

Guess the second queen

Guess the third queen



Try another guess!

8 queens

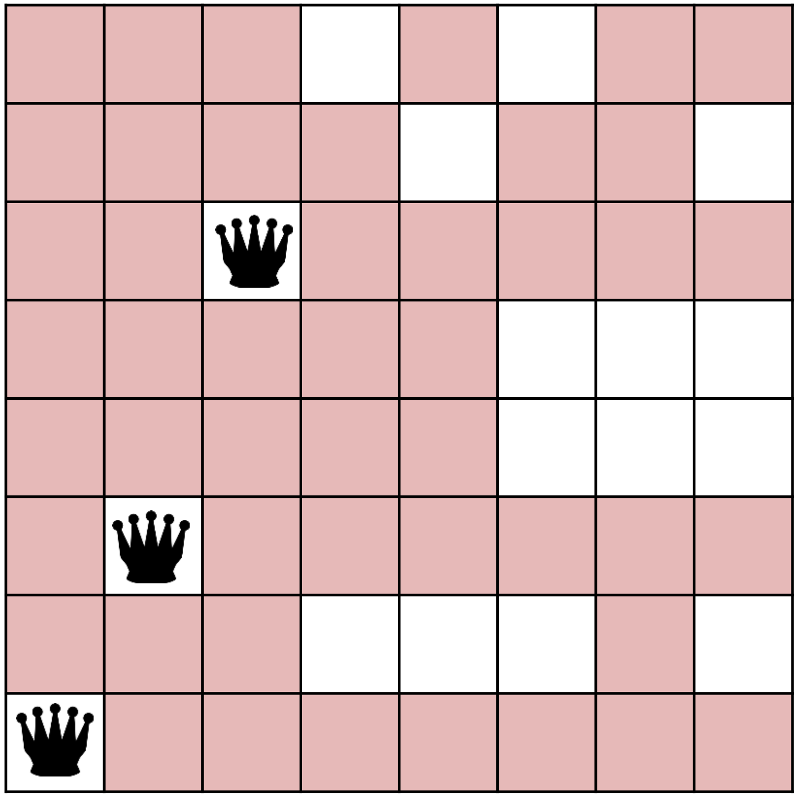
Task: place 8 queens on the chess board such that they do not attack each other

Guess the first queen

Guess the second queen

Guess the third queen

Apply constraints



Constraint programming

- Computational paradigm
  - use constraints to reduce the set of values that each variable can take
  - make a choice if no deduction can be made
- What is the choice?
  - there are many choices!
  - for the moment, assume a choice assigns a value to a variable
  - give solvers as much information as possible
- Choices can go wrong!
  - Try another one

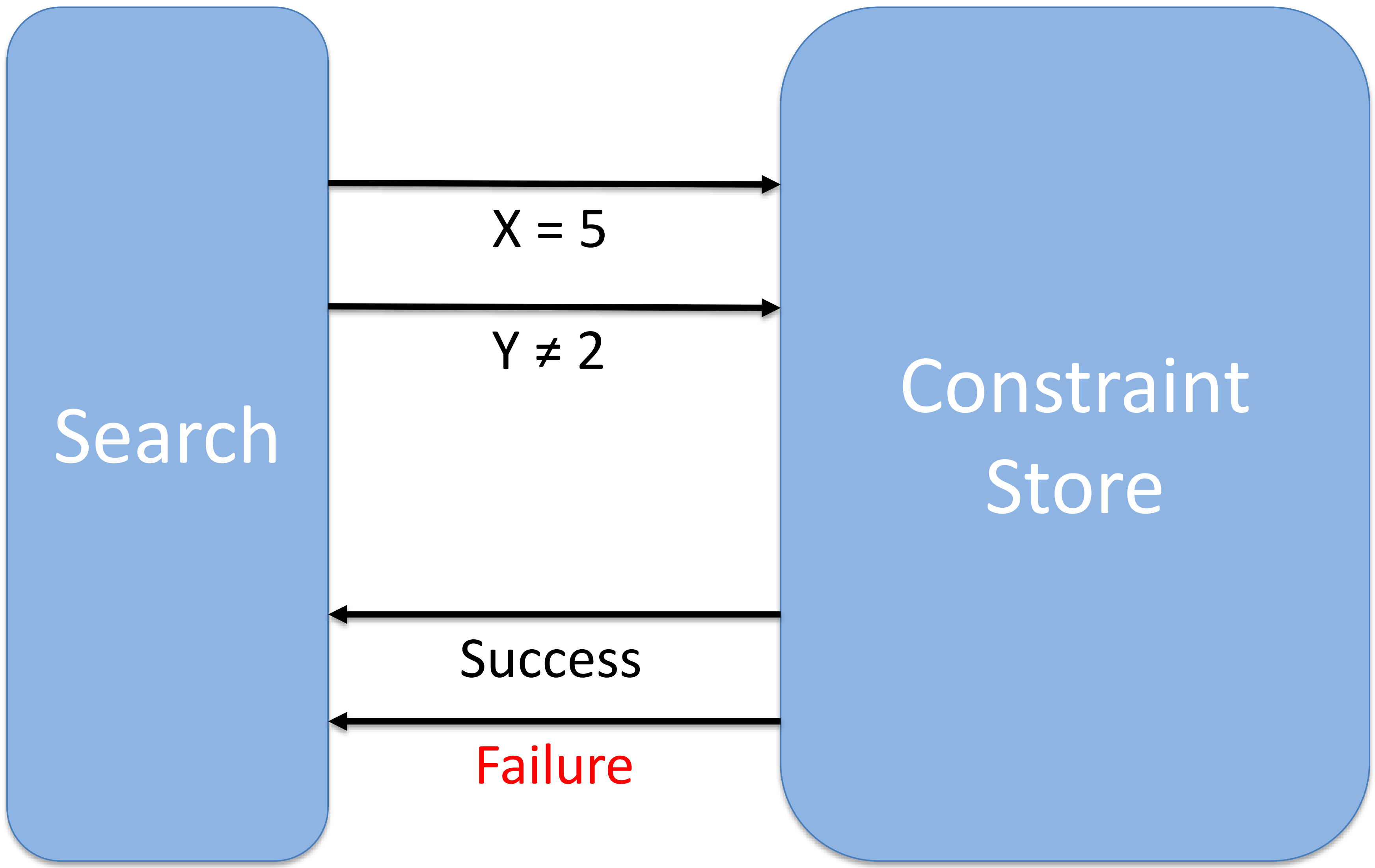
Computational paradigm

- Branch and prune
  - pruning: reduce the search space as much as possible
  - branching: decompose the problem into subproblems and explore the subproblems
- Pruning
  - use constraints to remove values that cannot belong to any solution from the variable domains
- Branching
  - try all the possible values of a variable until a solution is found or it can be proven that no solution exists

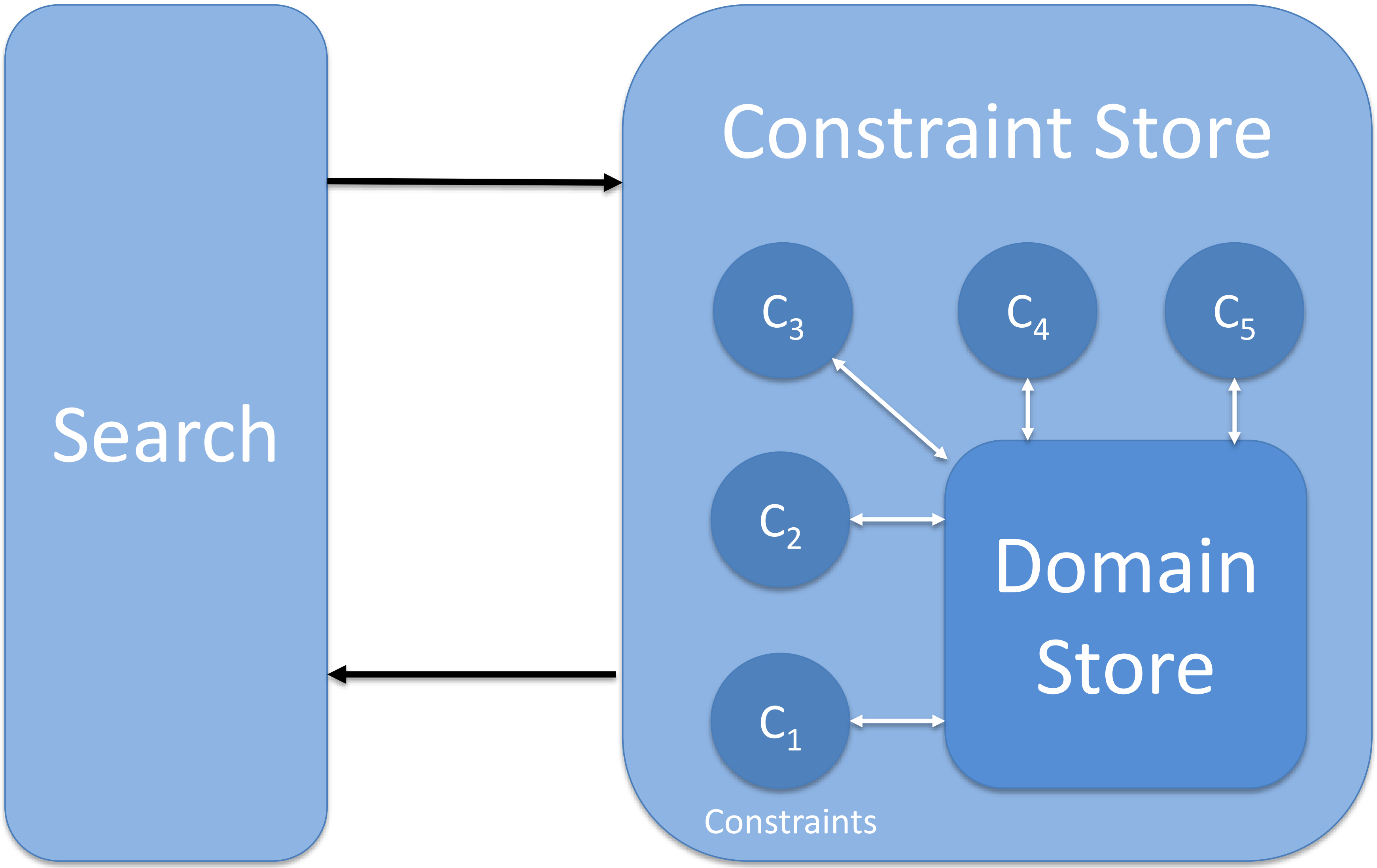
Computational paradigm

- Complete method, not a heuristic
  - given enough time, it will find a solution to a satisfaction problem
- Focus on feasibility
  - how to use constraints to prune the search space by eliminating values that cannot belong to any solution

Computational paradigm



Computational paradigm



Computational paradigm

- What does a constraint do?
  - feasibility checking
  - pruning
- Feasibility checking
  - a constraint checks if it can be satisfied given the values in the domains of its variables
- Branching
  - if satisfiable, a constraint determines which values in the domains cannot be part of any solution

Computational paradigm

- Propagation engine
  - the core of any constraint programming solver
  - a simple iterative algorithm to reach a stable state

```
propagate()
{
  repeat
    select a constraint c;
    if c is infeasible given the domain store then
      return failure;
    else
      apply the pruning algorithm associated with c;
  until no constraint can remove any value from the
  domain of its variables;
  return success;
}
```

8-queen problem

Task: place 8 queens on the chess board such that they do not attack each other

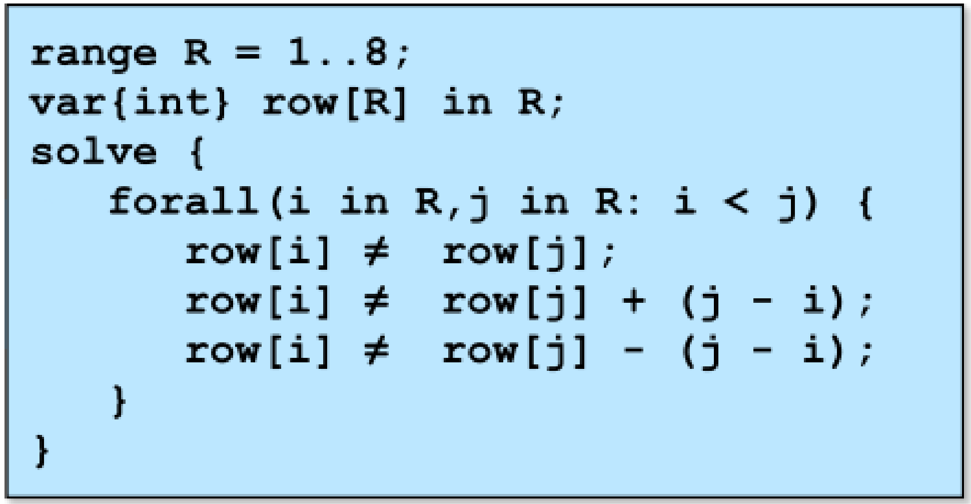
- Many ways to model
- Associate a decision variable with each column
  - the variable denotes the row of the queen in that column
  - no two queens can be placed on the same column
- What are the constraints?
  - the queens cannot be placed on the same
    - row
    - upward diagonal
    - downward diagonal

8-queen problem

Task: place 8 queens on the chess board such that they do not attack each other

Constraints: the queens cannot be placed on the same

- row
- upward diagonal
- downward diagonal



```
range R = 1..8;
var{int} row[R] in R;
solve {
   forall(i in R,j in R: i < j) {
      row[i] ≠  row[j];
      row[i] ≠  row[j] + (j - i);
      row[i] ≠  row[j] - (j - i);
   }
}
```

Computational paradigm

Consider two variables X and Y

Domains: $D(X) = \{0,1,2\}$, $D(Y) = \{1,2,3\}$

Consider constraint $X \neq Y$

Feasibility checking:

$|D(X) \cup D(Y)| \geq 2$

$|\{0,1,2,3\}| \geq 2$

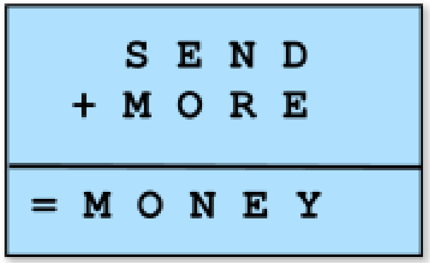
$4 \geq 2$

Pruning?
when variables take only one value

If $D(X) = \{1\}$
Then $D(Y) := D(Y) \setminus \{1\}$

More constraints: Send More Money

Task: assign different digits to letters to satisfy the addition



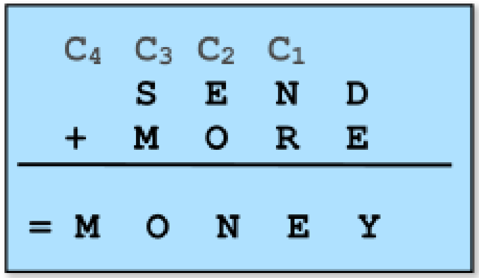
```
    S E N D
  + M O R E
-----------
= M O N E Y
```

What are the decision variables?

– there is a variable for each letter to denote the value

Send More Money

Task: assign different digits to letters to satisfy the addition



$$
\begin{array}{cccccc}
 & C_4 & C_3 & C_2 & C_1 & \\
 & & S & E & N & D \\
 & + & M & O & R & E \\
\hline
= & M & O & N & E & Y
\end{array}
$$

What are the decision variables?

– there is a variable for each letter to denote the value

– there is a variable for each carry

Send More Money

Task: assign different digits to letters to satisfy the addition

$$
\begin{array}{cccccc}
 & C_4 & C_3 & C_2 & C_1 & \\
 & & S & E & N & D \\
 & + & M & O & R & E \\
\hline
= & M & O & N & E & Y
\end{array}
$$

Constraints?

```
enum Letters = { S, E, N, D, M, O, R, Y};
range Digits = 0..9;
var{int} value[Letters] in Digits;
var{int} carry[1..4] in 0..1;

solve {
  forall(i in Letters, j in Letters: i < j)
     value[i] ≠ value[j];
  value[S] ≠ 0;
  value[M] ≠ 0;
  carry[4]                             = value[M];
  carry[3] + value[S] + value[M] = value[O] + 10 * carry[4];
  carry[2] + value[E] + value[O] = value[N] + 10 * carry[3];
  carry[1] + value[N] + value[R] = value[E] + 10 * carry[2];
             value[D] + value[E] = value[Y] + 10 * carry[1];
}
```

Send More Money

Task: assign different digits to letters to satisfy the addition

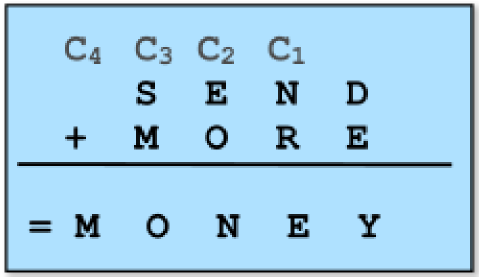
```
   C4  C3  C2  C1
       S   E   N   D
   +   M   O   R   E
  ------------------
  = M  O   N   E   Y
```

Search space?

	0	1	2	3	4	5	6	7	8	9
S										
E										
N										
D										
M										
O										
R										
Y										
C_4										
C_3										
C_2										
C_1										

Send More Money

Task: assign different digits to letters to satisfy the addition



$$\begin{array}{cccccc}
 & C_4 & C_3 & C_2 & C_1 & \\
 & & S & E & N & D \\
+ & & M & O & R & E \\
\hline
= & M & O & N & E & Y
\end{array}$$

value[S] ≠ 0

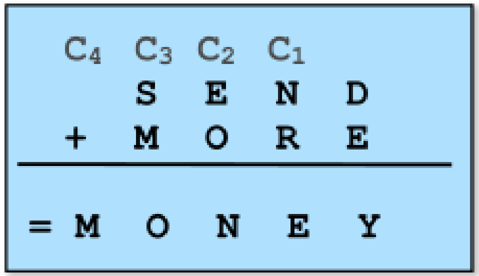
value[M] ≠ 0

carry[4] = value[M]

	0	1	2	3	4	5	6	7	8	9
S										
E										
N										
D										
M										
O										
R										
Y										
C_4										
C_3										
C_2										
C_1										

Send More Money

Task: assign different digits to letters to satisfy the addition



```
   C4 C3 C2 C1
       S  E  N  D
   +   M  O  R  E
   ---------------
   = M  O  N  E  Y
```

value[S] ≠ 0

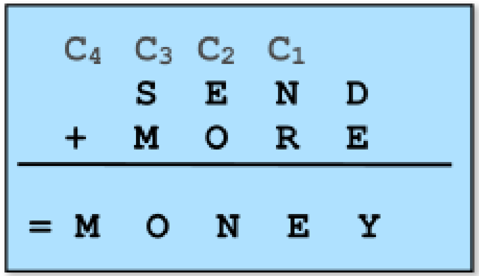
value[M] ≠ 0

carry[4] = value[M]

	0	1	2	3	4	5	6	7	8	9
S	🟥									
E										
N										
D										
M	🟥									
O										
R										
Y										
C_4		
C_3		
C_2		
C_1		

Send More Money

Task: assign different digits to letters to satisfy the addition



$$\begin{array}{cccccc} & C_4 & C_3 & C_2 & C_1 & \\ & & S & E & N & D \\ & + & M & O & R & E \\ \hline = & M & O & N & E & Y \end{array}$$

value[S] ≠ 0

value[M] ≠ 0

carry[4] = value[M]

	0	1	2	3	4	5	6	7	8	9
S	🟥									
E										
N										
D										
M	🟥									
O										
R										
Y										
C_4	🟥									
C_3										
C_2										
C_1										

Send More Money

Task: assign different digits to letters to satisfy the addition

$$\begin{array}{cccccc}
 & C_4 & C_3 & C_2 & C_1 & \\
 & & S & E & N & D \\
+ & & M & O & R & E \\
\hline
= & M & O & N & E & Y
\end{array}$$

value[S] ≠ 0

value[M] ≠ 0

carry[4] = value[M]

	0	1	2	3	4	5	6	7	8	9
S	🟥									
E										
N										
D										
M	🟥									
O										
R										
Y										
C_4	🟥	🟦								
C_3										
C_2										
C_1										

Send More Money

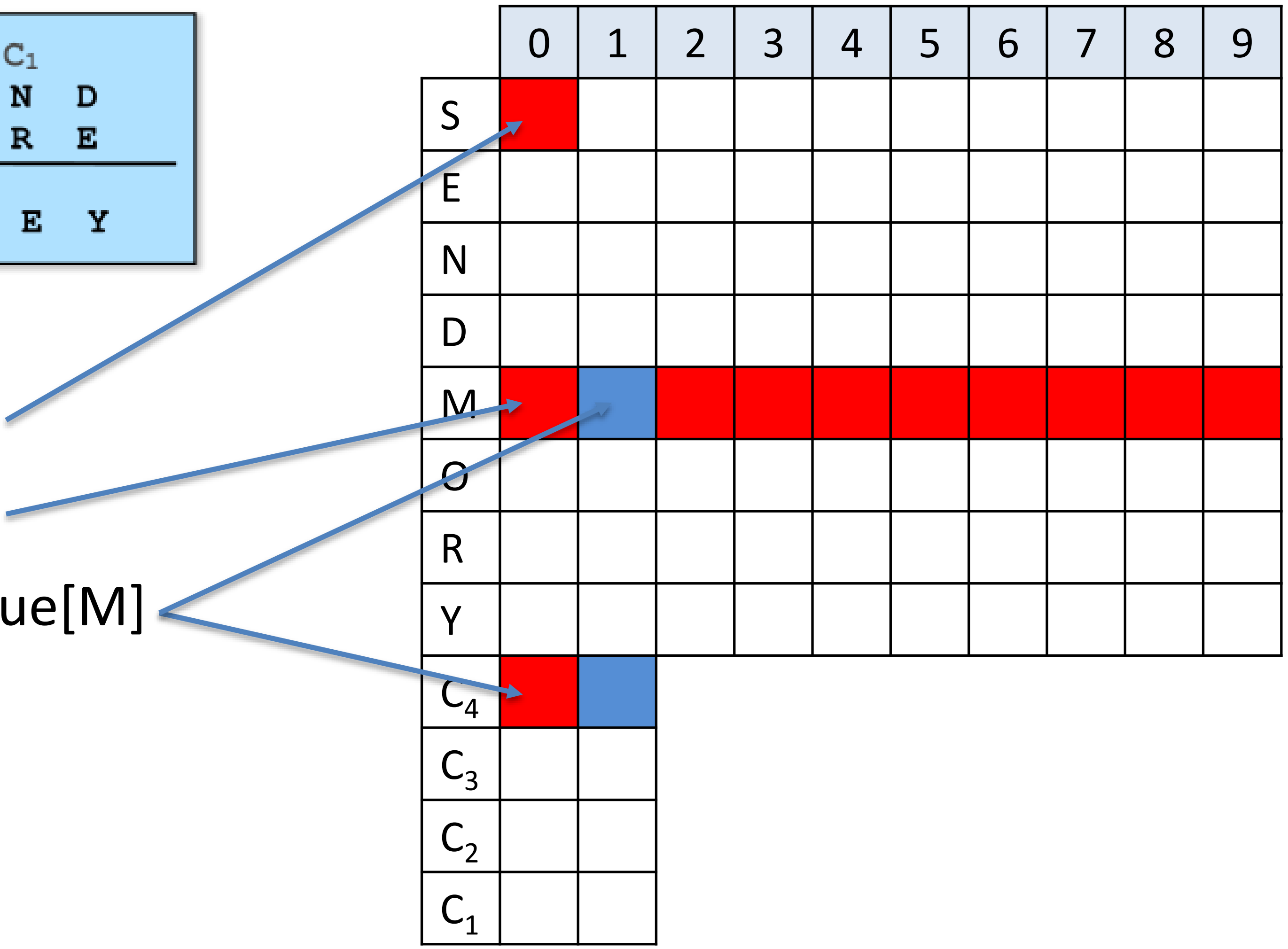
Task: assign different digits to letters to satisfy the addition

$$\begin{array}{cccccc}
 & C_4 & C_3 & C_2 & C_1 & \\
 & & S & E & N & D \\
+ & & M & O & R & E \\
\hline
= & M & O & N & E & Y
\end{array}$$

value[S] ≠ 0

value[M] ≠ 0

carry[4] = value[M]



Send More Money

Task: assign different digits to letters to satisfy the addition

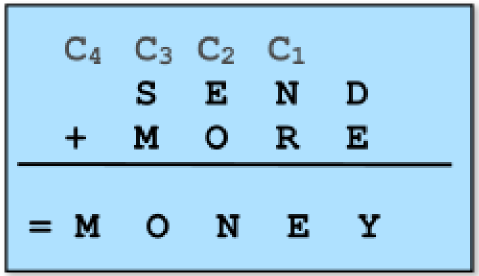
```
  C4  C3  C2  C1
      S   E   N   D
  +   M   O   R   E
 ------------------
 = M  O   N   E   Y
```

value[i] ≠ value[j]

	0	1	2	3	4	5	6	7	8	9
S	🟥									
E										
N										
D										
M	🟥	🟦	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
O										
R										
Y										
C_4	🟥	🟦								
C_3										
C_2										
C_1										

Send More Money

Task: assign different digits to letters to satisfy the addition



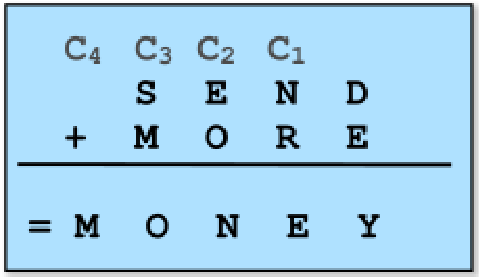
```
  C4  C3  C2  C1
      S   E   N   D
  +   M   O   R   E
  -----------------
= M   O   N   E   Y
```

value[i] ≠ value[j]

	0	1	2	3	4	5	6	7	8	9
S	🟥	🟥								
E		🟥								
N		🟥								
D		🟥								
M	🟥	🟦	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
O		🟥								
R		🟥								
Y		🟥								
C_4	🟥	🟦								
C_3										
C_2										
C_1										

Send More Money

Task: assign different digits to letters to satisfy the addition



```
   C4  C3  C2  C1
       S   E   N   D
   +   M   O   R   E
  ------------------
  = M  O   N   E   Y
```

carry[3]+value[S]+value[M]=

=value[O]+10*carry[4]

	0	1	2	3	4	5	6	7	8	9
S	🟥	🟥								
E		🟥								
N		🟥								
D		🟥								
M	🟥	🟦	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
O		🟥								
R		🟥								
Y		🟥								
C_4	🟥	🟦								
C_3										
C_2										
C_1										

Send More Money

Task: assign different digits to letters to satisfy the addition

```
   C4  C3  C2  C1
       S   E   N   D
   +   M   O   R   E
 ----------------------
 = M   O   N   E   Y
```

carry[3]+value[S]+1=

=value[O]+10*1

	0	1	2	3	4	5	6	7	8	9
S	🟥	🟥								
E		🟥								
N		🟥								
D		🟥								
M	🟥	🟦	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
O		🟥								
R		🟥								
Y		🟥								
C_4	🟥	🟦								
C_3										
C_2										
C_1										

Send More Money

Task: assign different digits to letters to satisfy the addition

$$
\begin{array}{cccccc}
 & C_4 & C_3 & C_2 & C_1 & \\
 & & S & E & N & D \\
+ & & M & O & R & E \\
\hline
= & M & O & N & E & Y
\end{array}
$$

carry[3]+value[S]+1=

=value[O]+10*1

lhs $\in$ [3, …, 11]

rhs $\in$ [10, …, 19]

=> lhs = rhs $\in$ [10, 11]

	0	1	2	3	4	5	6	7	8	9
S	🟥	🟥								
E		🟥								
N		🟥								
D		🟥								
M	🟥	🟦	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
O		🟥								
R		🟥								
Y		🟥								
C_4	🟥	🟦								
C_3										
C_2										
C_1										

Send More Money

Task: assign different digits to letters to satisfy the addition

$$\begin{array}{cccccc}
 & C_4 & C_3 & C_2 & C_1 & \\
 & & S & E & N & D \\
+ & & M & O & R & E \\
\hline
= & M & O & N & E & Y
\end{array}$$

carry[3]+value[S]+1=

=value[O]+10 $\in$ [10, 11]

=>

9 $\leq$ carry[3]+value[S] $\leq$ 10

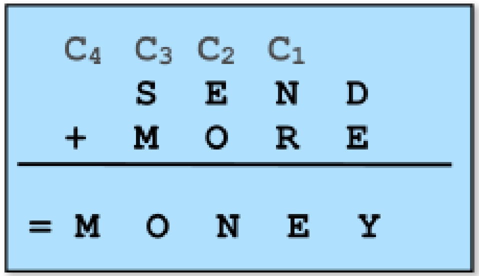
=>

8 $\leq$ value[S] $\leq$ 10

	0	1	2	3	4	5	6	7	8	9
S	🟥	🟥								
E		🟥								
N		🟥								
D		🟥								
M	🟥	🟦	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
O		🟥								
R		🟥								
Y		🟥								
C_4	🟥	🟦								
C_3										
C_2										
C_1										

Send More Money

Task: assign different digits to letters to satisfy the addition



```
  C4  C3  C2  C1
      S   E   N   D
  +   M   O   R   E
  -----------------
  = M   O   N   E   Y
```

carry[3]+value[S]+1=

=value[O]+10 $\in$ [10, 11]

=>

9 ≤ carry[3]+value[S] ≤ 10

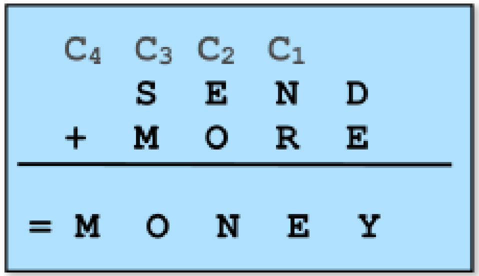
=>

8 ≤ value[S] ≤ 10

	0	1	2	3	4	5	6	7	8	9
S	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥		
E		🟥								
N		🟥								
D		🟥								
M	🟥	🟦	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
O		🟥								
R		🟥								
Y		🟥								
C_4	🟥	🟦								
C_3										
C_2										
C_1										

Send More Money

Task: assign different digits to letters to satisfy the addition



$$\begin{array}{cccccc}
 & C_4 & C_3 & C_2 & C_1 & \\
 & & S & E & N & D \\
+ & & M & O & R & E \\
\hline
= & M & O & N & E & Y
\end{array}$$

carry[3] + value[S] + 1

= value[O] + 10 *carry[4]

rhs ∈ [10, 19]

lhs ∈ [8, 11]

=>

rhs = lhs ∈ [10, 11]

	0	1	2	3	4	5	6	7	8	9
S	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥		
E		🟥								
N		🟥								
D		🟥								
M	🟥	🟦	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
O		🟥								
R		🟥								
Y		🟥								
C_4	🟥	🟦								
C_3										
C_2										
C_1										

Send More Money

Task: assign different digits to letters to satisfy the addition

$$\begin{array}{cccccc}
 & C_4 & C_3 & C_2 & C_1 & \\
 & & S & E & N & D \\
+ & & M & O & R & E \\
\hline
= & M & O & N & E & Y
\end{array}$$

value[O] $\in$ [0, 1]

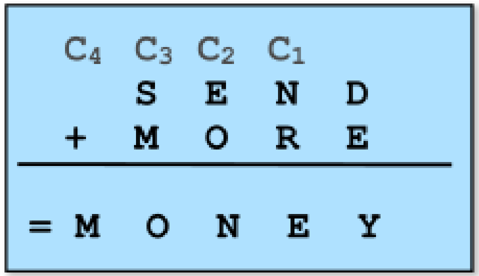
=>

value[O] = 0

	0	1	2	3	4	5	6	7	8	9
S	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥		
E		🟥								
N		🟥								
D		🟥								
M	🟥	🟦	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
O		🟥								
R		🟥								
Y		🟥								
C_4	🟥	🟦								
C_3										
C_2										
C_1										

Send More Money

Task: assign different digits to letters to satisfy the addition



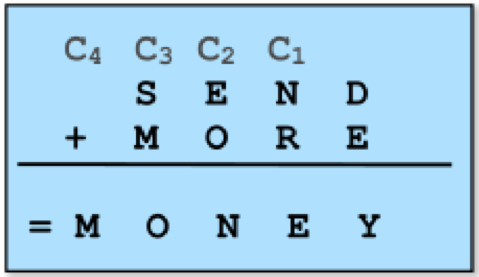
```
   C4  C3  C2  C1
       S   E   N   D
   +   M   O   R   E
 ---------------------
 = M   O   N   E   Y
```

value[i] ≠ value[j]

	0	1	2	3	4	5	6	7	8	9
S	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥		
E		🟥								
N		🟥								
D		🟥								
M	🟥	🟦	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
O	🟦	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
R		🟥								
Y		🟥								
C_4	🟥	🟦								
C_3										
C_2										
C_1										

Send More Money

Task: assign different digits to letters to satisfy the addition



```
   C4  C3  C2  C1
       S   E   N   D
   +   M   O   R   E
  ------------------
 = M   O   N   E   Y
```

value[i] ≠ value[j]

	0	1	2	3	4	5	6	7	8	9
S	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥		
E	🟥	🟥								
N	🟥	🟥								
D	🟥	🟥								
M	🟥	🟦	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
O	🟦	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
R	🟥	🟥								
Y	🟥	🟥								
C_4	🟥	🟦								
C_3										
C_2										
C_1										

Send More Money

Task: assign different digits to letters to satisfy the addition

```
  C4  C3  C2  C1
      S   E   N   D
  +   M   O   R   E
 ------------------
= M   O   N   E   Y
```

carry[2]+value[E]+value[O]

=value[N]+10*carry[3]

	0	1	2	3	4	5	6	7	8	9
S	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥		
E	🟥	🟥								
N	🟥	🟥								
D	🟥	🟥								
M	🟥	🟦	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
O	🟦	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
R	🟥	🟥								
Y	🟥	🟥								
C_4	🟥	🟦								
C_3										
C_2										
C_1										

Send More Money

Task: assign different digits to letters to satisfy the addition

```
   C4  C3  C2  C1
       S   E   N   D
   +   M   O   R   E
   ----------------
= M    O   N   E   Y
```

carry[2]+value[E]

=value[N]+10*carry[3]

lhs ∈ [2, 10]

=>

carry[3] = 0

	0	1	2	3	4	5	6	7	8	9
S	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥		
E	🟥	🟥								
N	🟥	🟥								
D	🟥	🟥								
M	🟥	🟦	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
O	🟦	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
R	🟥	🟥								
Y	🟥	🟥								
C_4	🟥	🟦								
C_3										
C_2										
C_1										

Send More Money

Task: assign different digits to letters to satisfy the addition

```
  C4  C3  C2  C1
      S   E   N   D
  +   M   O   R   E
  -----------------
= M   O   N   E   Y
```

carry[2]+value[E]

=value[N]+10*carry[3]

lhs ∈ [2, 10]

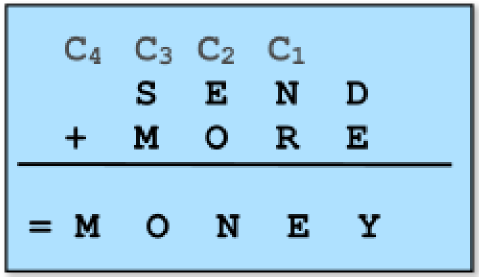
=>

carry[3] = 0

	0	1	2	3	4	5	6	7	8	9
S	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥		
E	🟥	🟥								
N	🟥	🟥								
D	🟥	🟥								
M	🟥	🟦	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
O	🟦	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
R	🟥	🟥								
Y	🟥	🟥								
C_4	🟥	🟦								
C_3	🟦	🟥								
C_2										
C_1										

Send More Money

Task: assign different digits to letters to satisfy the addition



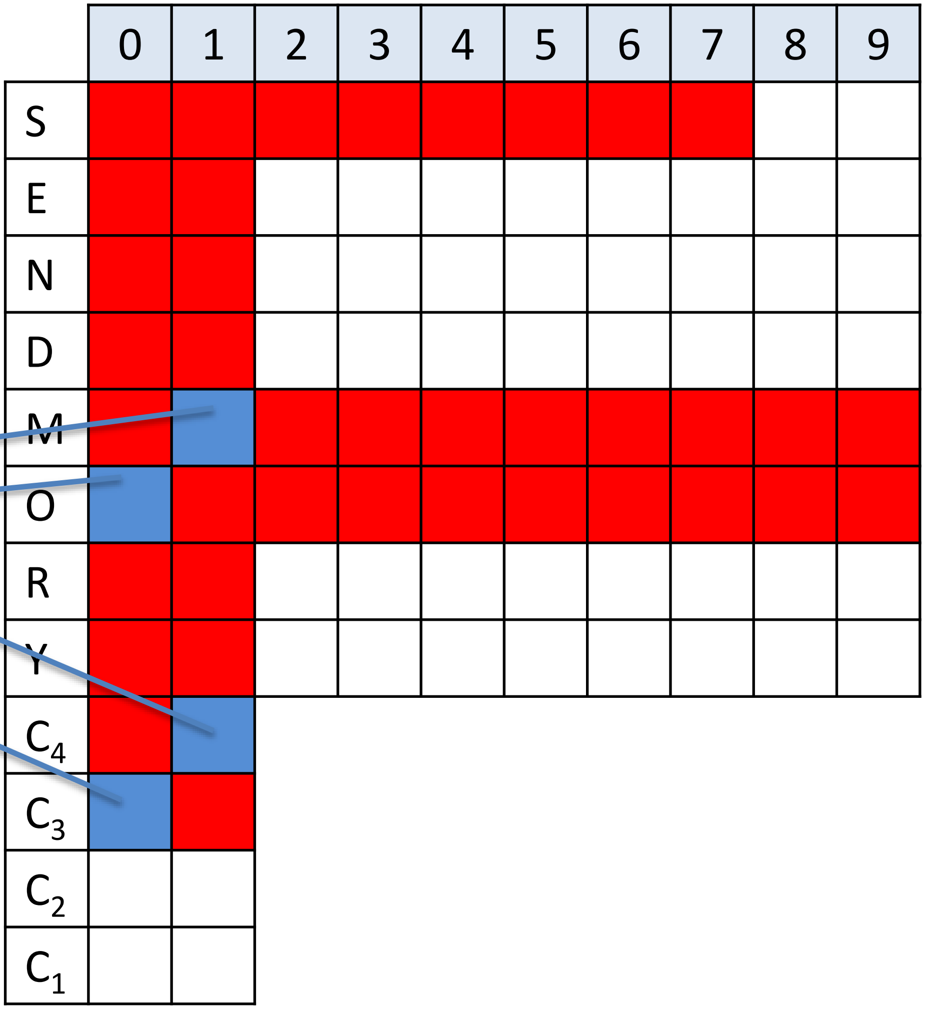
```
   C4 C3 C2 C1
      S  E  N  D
   +  M  O  R  E
   -------------
   = M  O  N  E  Y
```

carry[3]+value[S]+value[M]

=value[O]+10*carry[4]

=>

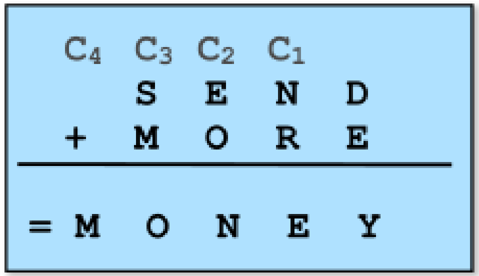
value[S] = 9



	0	1	2	3	4	5	6	7	8	9
S	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥		
E	🟥	🟥								
N	🟥	🟥								
D	🟥	🟥								
M	🟥	🟦	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
O	🟦	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
R	🟥	🟥								
Y	🟥	🟥								
C_4	🟥	🟦
C_3	🟦	🟥
C_2		
C_1		

Send More Money

Task: assign different digits to letters to satisfy the addition



$$\begin{array}{cccccc} & C_4 & C_3 & C_2 & C_1 & \\ & & S & E & N & D \\ + & & M & O & R & E \\ \hline = & M & O & N & E & Y \end{array}$$

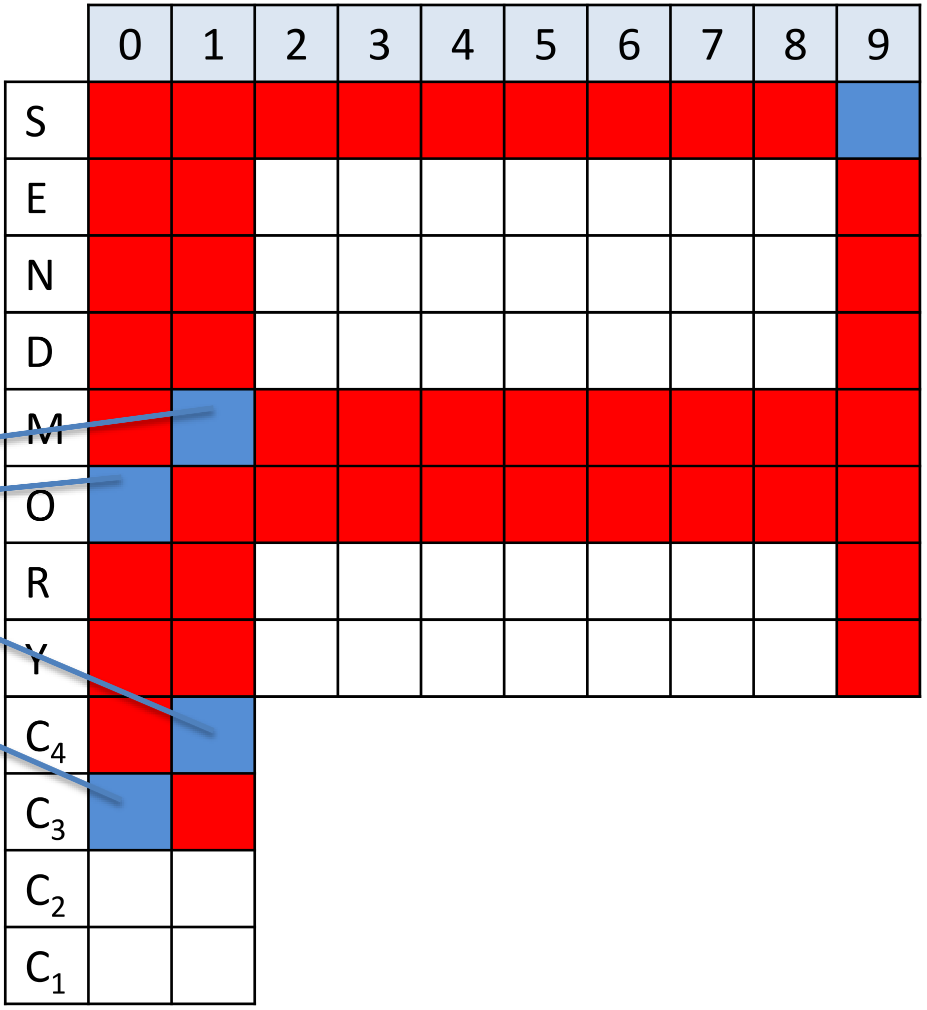
carry[3]+value[S]+value[M]

=value[O]+10*carry[4]

=>

value[S] = 9

value[i] ≠ value[j]



	0	1	2	3	4	5	6	7	8	9
S	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟦
E	🟥	🟥								🟥
N	🟥	🟥								🟥
D	🟥	🟥								🟥
M	🟥	🟦	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
O	🟦	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
R	🟥	🟥								🟥
Y	🟥	🟥								🟥
C_4	🟥	🟦								
C_3	🟦	🟥								
C_2										
C_1										

Linear constraints over integers

Consider a constraint

$a_1\ x_1 + \ldots + a_n\ x_n \geq b_1\ y_1 + \ldots + b_m\ y_m$

a_i, $b_j \geq$ are constants

x_i, y_j are variables with domains $D(x_i)$, $D(y_j)$

Feasibility test:

$a_1 \max(D(x_1)) + \ldots + a_n \max(D(x_n)) \geq b_1 \min(D(y_1)) + \ldots + b_m \min(D(y_m))$

Pruning:

$a_i\ x_i \geq B - (\ A - a_i \max(D(x_i))\ )$

$b_j\ y_j \leq A - (\ B - b_j \max(D(y_j))\ )$

Symmetry breaking

- Many problems naturally exhibit symmetries
  - Exploring symmetrical parts of the search space is useless
- Many kinds of symmetries
  - Variable symmetries
  - Value symmetries
- Symmetry breaking constraints

Symmetry breaking: variable symmetries

- Balanced Incomplete Block Designs (BIBDs)
  - Input: (v, b, r, k, l)
  - Output: v × b matrix of 0/1 with exactly r ones per row, k ones per column, and a scalar product of rows is l

- Why BIBDs?
  - Example of combinatorial design
  - Full of variable symmetries

(3, 3, 2, 2, 1)

1	1	0
0	1	1
1	0	1

Symmetry breaking: BIBDs

- Balanced Incomplete Block Designs (BIBDs)
  - Input: (v, b, r, k, l)
  - Output: v × b matrix of 0/1 with exactly r ones per row, k ones per column, and a scalar product of rows is l

```
range Rows = 1..v;
range Cols = 1..b;
var{int} m[Rows,Cols] in 0..1;
solve {
   forall(i in Rows)
      sum(y in Cols) m[i,y] = r;
   forall(j in Cols)
      sum(x in Rows) m[x,j] = k;
   forall(i in Rows,j in Rows: j > i)
      sum(x in Cols) (m[i,x] & m[j,x]) = 1;
}
```

(3, 3, 2, 2, 1)

1	1	0
0	1	1
1	0	1

Symmetry breaking: BIBDs

(7, 7, 3, 3, 1)

0	1	1	0	0	1	0
1	0	1	0	1	0	0
0	0	1	1	0	0	1
1	1	0	0	0	0	1
0	0	0	0	1	1	1
1	0	0	1	0	1	0
0	1	0	1	1	0	0

Swapping rows

1	0	1	0	1	0	0
0	1	1	0	0	1	0
0	0	1	1	0	0	1
1	1	0	0	0	0	1
0	0	0	0	1	1	1
1	0	0	1	0	1	0
0	1	0	1	1	0	0

Symmetry breaking: BIBDs

(7, 7, 3, 3, 1)

0	1	1	0	0	1	0
1	0	1	0	1	0	0
0	0	1	1	0	0	1
1	1	0	0	0	0	1
0	0	0	0	1	1	1
1	0	0	1	0	1	0
0	1	0	1	1	0	0

Swapping columns

0	1	1	0	0	1	0
1	0	1	0	1	0	0
0	0	1	1	0	0	1
1	0	0	0	0	1	1
0	1	0	0	1	0	1
1	1	0	1	0	0	0
0	0	0	1	1	1	0

Symmetry breaking: BIBDS

- How to break variable symmetries
  - Impose an ordering on the variables
- Consider the row symmetries
  - Impose a lexicographic constraint
- Lexicographic ordering

a :	0 1 1 0 0 1 0	1 1 1 0 0 1 0
b :	1 0 1 0 1 0 0	1 0 1 0 1 0 0
	$a \leq b$	$a \geq b$

Symmetry breaking: BIBDs

(7, 7, 3, 3, 1)

0	1	1	0	0	1	0
1	0	1	0	1	0	0
0	0	1	1	0	0	1
1	1	0	0	0	0	1
0	0	0	0	1	1	1
1	0	0	1	0	1	0
0	1	0	1	1	0	0

Lexicographic Ordering

0	0	0	0	1	1	1
0	0	1	1	0	0	1
0	1	0	1	1	0	0
0	1	1	0	0	1	0
1	0	0	1	0	1	0
1	0	1	0	1	0	0
1	1	0	0	0	0	1

Symmetry breaking: BIBDs

(7, 7, 3, 3, 1)

0	0	0	0	1	1	1
0	0	1	1	0	0	1
0	1	0	1	1	0	0
0	1	1	0	0	1	0
1	0	0	1	0	1	0
1	0	1	0	1	0	0
1	1	0	0	0	0	1

Break column symmetries

0	0	0	0	1	1	1
0	0	1	1	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	1	0	0	0	0	1

Symmetry breaking: BIBDs

```
range Rows = 1..v;
range Cols = 1..b;
var{int} m[Rows,Cols] in 0..1;
solve {
   forall(i in Rows)
      sum(y in Cols) m[i,y] = r;
   forall(j in Cols)
      sum(x in Rows) m[x,j] = k;
   forall(i in Rows,j in Rows: j > i)
      sum(x in Cols) (m[i,x] & m[j,x]) = 1;
   forall(i in 1..v-1)
      lexleq(all(j in Cols) m[i,j],all(j in Cols) m[i+1,j]);
   forall(j in 1..b-1)
      lexleq(all(i in Rows) m[i,j],all(i in Rows) m[i,j+1]);
}
```

Symmetry breaking: value symmetries

Scene allocation problem

- Shooting scenes for a movie
  - an actor plays in some of the scenes
  - at most k scenes can be shot per day
  - each actor is paid by the day
  - scenes are different, actors are different
- Objective
  - Minimize the total cost
- Symmetries
  - What kind of symmetries do we have here?

Symmetry breaking: scene allocation

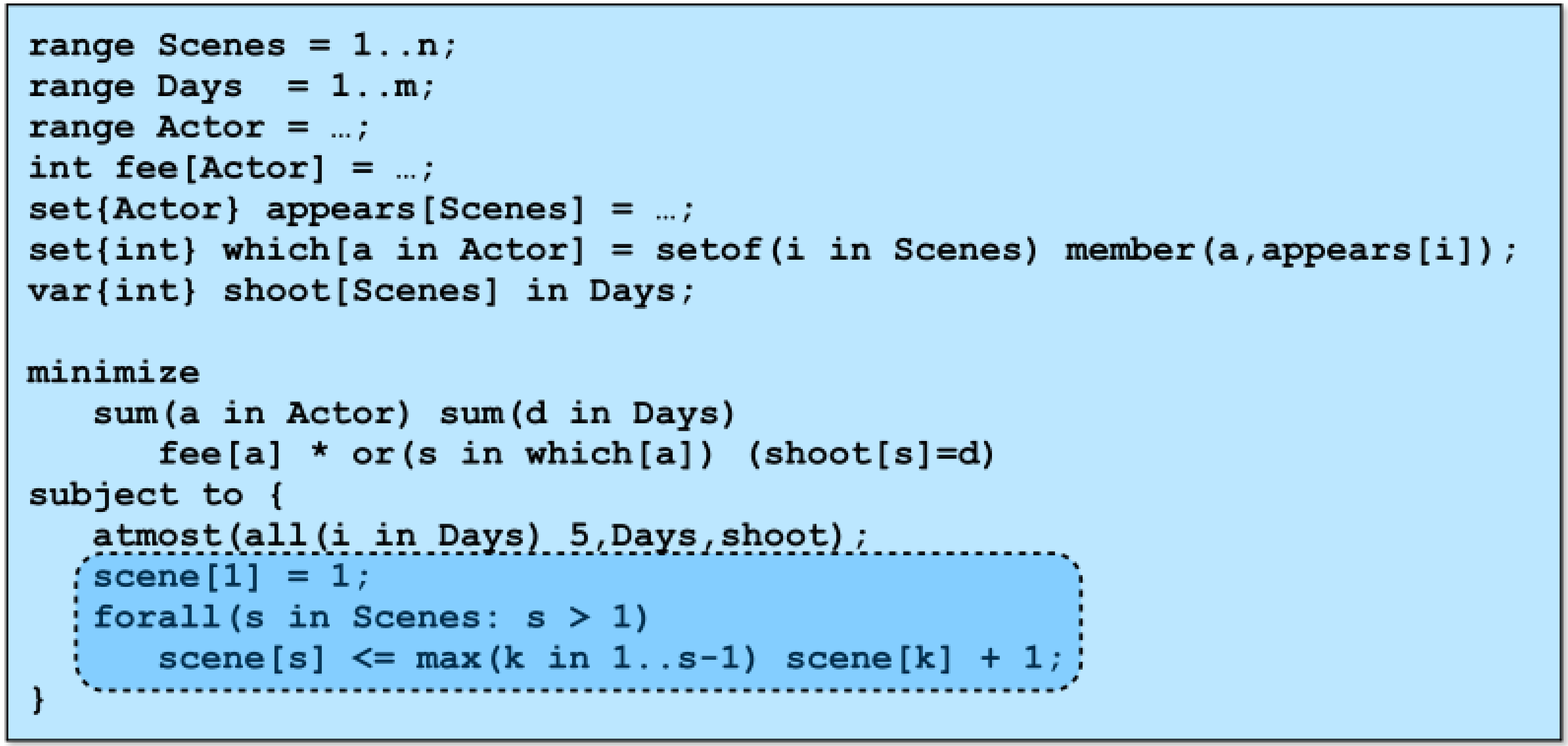
```
range Scenes = …;
range Days   = …;
range Actor = …;
int fee[Actor] = …;
set{Actor} appears[Scenes] = …;
set{int} which[a in Actor] = setof(i in Scenes) member(a,appears[i]);
var{int} shoot[Scenes] in Days;

minimize
   sum(a in Actor) sum(d in Days)
      fee[a] * or(s in which[a]) (shoot[s]=d)
subject to
   atmost(all(i in Days) 5,Days,shoot);
```

Symmetry breaking: scene allocation

- Value symmetries
  - the days are interchangeable
  - can swap all the scenes in day 1 and all the scenes in day 2 and still have a solution
  - if s is a solution, then p(s) is a solution where the days of s have been permuted by permutation p
- How do we eliminate these symmetries?
  - Consider the scene 1. What are the days that we consider for this scene?
  - Only day 1 for scene 1.
  - Where do we schedule the second scene?
  - Day 1 or day 2.

Symmetry breaking: scene allocation

- How do we eliminate these symmetries?
  - Choose between the days already used and one new day.



```
range Scenes = 1..n;
range Days   = 1..m;
range Actor = …;
int fee[Actor] = …;
set{Actor} appears[Scenes] = …;
set{int} which[a in Actor] = setof(i in Scenes) member(a,appears[i]);
var{int} shoot[Scenes] in Days;

minimize
   sum(a in Actor) sum(d in Days)
      fee[a] * or(s in which[a]) (shoot[s]=d)
subject to {
   atmost(all(i in Days) 5,Days,shoot);
   scene[1] = 1;
   forall(s in Scenes: s > 1)
      scene[s] <= max(k in 1..s-1) scene[k] + 1;
}
```

Optimization in constraint programming?

- Focus of constraint programming
  - Feasibility

- How to optimize?
  - Solve a sequence of satisfaction problems
  - Find a solution
  - Impose a constraint that the new solution mush be better
- Guaranteed to find an optimal solution
  - at least theoretically
  - Strong when the new constraint reduces the search space
  - Works well for scheduling problems

Redundant constraints

- Motivation
  - Semantically redundant (do not exclude any solution)
  - Computationally significant (reduce the search space)
- How do we find redundant constraints?
  - they express properties of the solutions not captures by the model
- Critical aspect of constraint programming!

Redundant constraints: magic series

A series $S = (S_0, \ldots, S_n)$ is magic if S_i represents the number of occurrences of i in S

	0	1	2	3	4
Occurences	?	?	?	?	?

Redundant constraints: magic series

A series S = (S_0, …, S_n) is magic if S_i represents the number of occurrences of i in S

	0	1	2	3	4
Occurences	2	1	2	0	0

How to find magic series?

```
int n = 5;
range D = 0..n-1;
var{int} series[D] in D;
solve {
   forall(k in D)
     series[k] = sum(i in D) (series[i]=k);
}
```

Redundant constraints: magic series

A series S = (S_0, ..., S_n) is magic if S_i represents the number of occurrences of i in S

```
int n = 5;
range D = 0..n-1;
var{int} series[D] in D;
solve {
   forall(k in D)
     series[k] = sum(i in D) (series[i]=k);
}
```

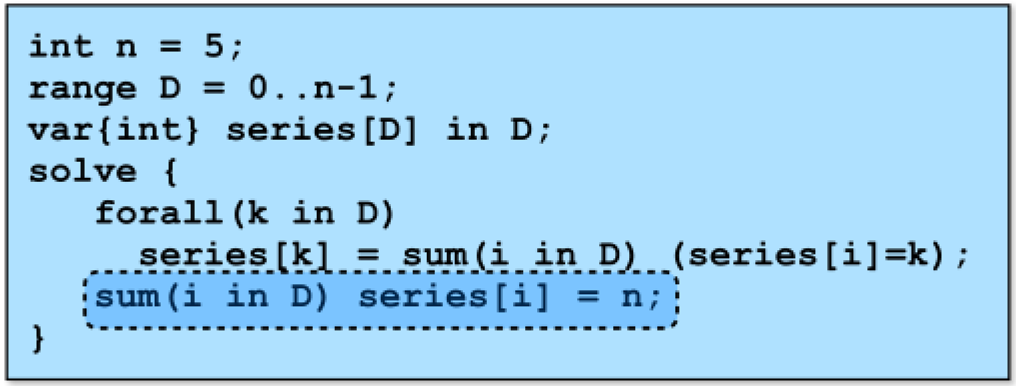
Redundant constraints:

	0	1	2	3	4
Occurences	?	?	?	?	17

- The decision variables denote a number of occurrences
- The number of occurrences is bounded

Redundant constraints: magic series

A series $S = (S_0, \ldots, S_n)$ is magic if S_i represents the number of occurrences of i in S



```
int n = 5;
range D = 0..n-1;
var{int} series[D] in D;
solve {
   forall(k in D)
     series[k] = sum(i in D) (series[i]=k);
   sum(i in D) series[i] = n;
}
```

Redundant constraints: magic series

A series S = (S_0, …, S_n) is magic if S_i represents the number of occurrences of i in S

What does “series[2]=3” mean?

	0	1	2	3	4
Occurences	?	?	3	?	?

That there are three “2” in the array “series”

Constraint: $$\sum_{i=0}^{n} i \cdot \text{series}[i] = n$$

Redundant constraints: magic series

A series S = (S_0, …, S_n) is magic if S_i represents the number of occurrences of i in S

```
int n = 5;
range D = 0..n-1;
var{int} series[D] in D;
solve {
   forall(k in D)
     series[k] = sum(i in D) (series[i]=k);
   sum(i in D) series[i] = n;
   sum(i in D) i * series[i] = n;
}
```

Redundant constraints: magic series

A series $S = (S_0, \ldots, S_n)$ is magic if S_i represents the number of occurrences of i in S

```
series[0] = (series[0]=0)+(series[1]=0)+(series[2]=0)+(series[3]=0)+(series[4]=0);
series[1] = (series[0]=1)+(series[1]=1)+(series[2]=1)+(series[3]=1)+(series[4]=1);
series[2] = (series[0]=2)+(series[1]=2)+(series[2]=2)+(series[3]=2)+(series[4]=2);
series[3] = (series[0]=3)+(series[1]=3)+(series[2]=3)+(series[3]=3)+(series[4]=3);
series[4] = (series[0]=4)+(series[1]=4)+(series[2]=4)+(series[3]=4)+(series[4]=4);
series[1] + 2 series[2] + 3 series[3] + 4 series[4] = 5
```

The redundant constraint implies:

- series[4] ≤ 1
- series[3] ≤ 1
- series[2] ≤ 2
- series[1] ≤ 5

Redundant constraints: magic series

A series $S = (S_0, \ldots, S_n)$ is magic if S_i represents the number of occurrences of i in S

Choice: assume that series[0] = 2

```
2          =    (series[1]=0)+(series[2]=0)+(series[3]=0)+(series[4]=0);
series[1]  =    (series[1]=1)+(series[2]=1)+(series[3]=1)+(series[4]=1);
series[2]  = 1 +(series[1]=2)+(series[2]=2);
series[3]  =    (series[1]=3);
series[4]  =    (series[1]=4);
series[1]  + 2 series[2] + 3 series[3] + 4 series[4] = 5
```

It follows that series[2] ≥ 1

series[1] + 3 series[3] + 4 series[4] ≤ 4

series[4] ≤ 0; series[3] ≤ 1

Redundant constraints

- First role
  - express properties of the solutions
  - boost the propagation of other constraints
- Second role
  - provide a more global view
  - combine existing constraints
  - improve communication

Global constraints

- Critical feature of constraint programming
    - Capture combinatorial substructures arising in may applications
- Modeling
    - Make modeling easier and more natural
- Problem solving
    - Convey the problem structure to the solver that does not have to rediscover it
    - Give the ability to exploit dedicated algorithms

Global constraints: alldifferent

alldifferent(x_1,…,x_n)

specifies that x_1,…,x_n take values that are different

8 queens:

```
range R = 1..8;
var{int} row[R] in R;
solve {
   forall(i in R,j in R: i < j) {
      row[i] ≠  row[j];
      row[i] ≠  row[j] + (j - i);
      row[i] ≠  row[j] - (j - i);
   }
}
```

```
range R = 1..8;
var{int} row[R] in R;
solve {
   alldifferent(row);
   alldifferent(all(i in R) row[i]+i);
   alldifferent(all(i in R) row[i]-i);
}
```

Global constraints: all different

alldifferent($x_1,\ldots,x_n$)

specifies that $x_1,\ldots,x_n$ take values that are different

Constraint $c(x_1,\ldots,x_n)$ where $x_1 \in D_1 = D(x_1)$, $x_n \in D_n = D(x_n)$

Feasibility testing:

find values in the variable domains such that the constraint holds

$$\exists\, v_1 \in D_1, \ldots, v_n \in D_n : \quad c(x_1 = v_1, \ldots, x_n = v_n) = \text{true}$$

Global constraints: all different

Example:

a constraint alldifferent(x_1, x_2, x_3)

$x_1 \in \{1, 2\}$, $x_2 \in \{1, 2\}$, $x_3 \in \{1, 2\}$

Is this feasible?

No, only two values for 3 variables

(pigeon hole principle)

Each of the local constraints

$x_1 \neq x_2$, $x_2 \neq x_3$, $x_3 \neq x_1$

can be satisfied

Global constraints: all different

alldifferent($x_1,\dots,x_n$)

specifies that $x_1,\dots,x_n$ take values that are different

Constraint c($x_1,\dots,x_n$) where $x_1 \in D_1=D(x_1)$, $x_n \in D_n=D(x_n)$

Pruning

given v_i in D_i, does there exist a solution such that $x_i=v_i$?

For each value search for the values of variables such that the constraint holds

Global constraints: alldifferent

Example:

a constraint alldifferent(x_1, x_2, x_3)

$x_1 \in \{1, 2\}$, $x_2 \in \{1, 2\}$, $x_3 \in \{1, 2, 3\}$

Pruning?

$x_3 \neq 1$, $x_3 \neq 2$ => $D(x_3) = \{3\}$

Local constraints $x_1 \neq x_2$, $x_2 \neq x_3$, $x_3 \neq x_1$ do not produce pruning

Global constraints

- Global constraints deal with many variables at the same time
- Global constraints make it possible to discover infeasibilities earlier
- Global constraints make it possible to prune the search space more

Global constraints

Million-dollar question:

Can we detect feasibility and prune global constraints efficiently?

It depends on the constraints

- Sometimes we can
- Sometimes we need to relax standards
    - The pruning may be suboptimal
    - The pruning make take exponential time

Example: sudoku

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

Example: sudoku

```
range R = 1..9;
var{int} s[R,R] in R;
solve {
 //constraints on fixed positions
 forall(i in R)
   alldifferent(all(j in R) s[i,j]);
 forall(j in R)
   alldifferent(all(i in R) s[i,j]);
 forall(i in 0..2,j in 0..2)
   alldifferent(all(r in i*3+1..i*3+3,
                    c in j*3+1..j*3+3) s[r,c]);
}
```

Example: sudoku

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

Example: sudoku

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

Example: sudoku

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

Example: sudoku

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

Example: sudoku

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

Example: sudoku

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

Example: sudoku

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

Global constraints: table constraints

The simplest global constraint

Example: $X \in \{1, 2\}$, $Y \in \{1, 2\}$, $Z \in \{3, 4, 5\}$

Total possibilities: $|\{1,2\}| \times |\{1, 2\}| \times |\{3, 4, 5\}| = 12$

Table constraint	X	Y	Z
Combination 1	1	1	5
Combination 2	1	2	4
Combination 3	2	2	3
Combination 4	1	2	3

Global constraints: table constraints

The simplest global constraint

Example: $X \in \{1, 2\}$, $Y \in \{1, 2\}$, $Z \in \{3, 4, 5\}$

Total possibilities: $|\{1,2\}| \times |\{1, 2\}| \times |\{3, 4, 5\}| = 12$

Table constraint	X	Y	Z
~~Combination 1~~	~~1~~	~~1~~	~~5~~
Combination 2	1	2	4
Combination 3	2	2	3
Combination 4	1	2	3

Given $Z \neq 5$

=>

$Y = 2$

How to implement global constraints?

Two types of global constraints:

- knapsack
- alldifferent

Significant area of research:

- over 100 global constraints proposed so far

The Gold Standard for Pruning

- After pruning

  if value v is in the domain of variable x, then there exists a solution to the constraint with value v assigned to variable x

- Optimal pruning

  cannot prune more if only domains are considered

- Complexity

  in general, can’t be enforced in polynomial time

Binary knapsack

- The constraint

$$\ell \leq \sum_{k \in R} w_k x_k \leq u$$

$$x_k \in \{0,1\}$$

- Example

$$10 \leq 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 12$$

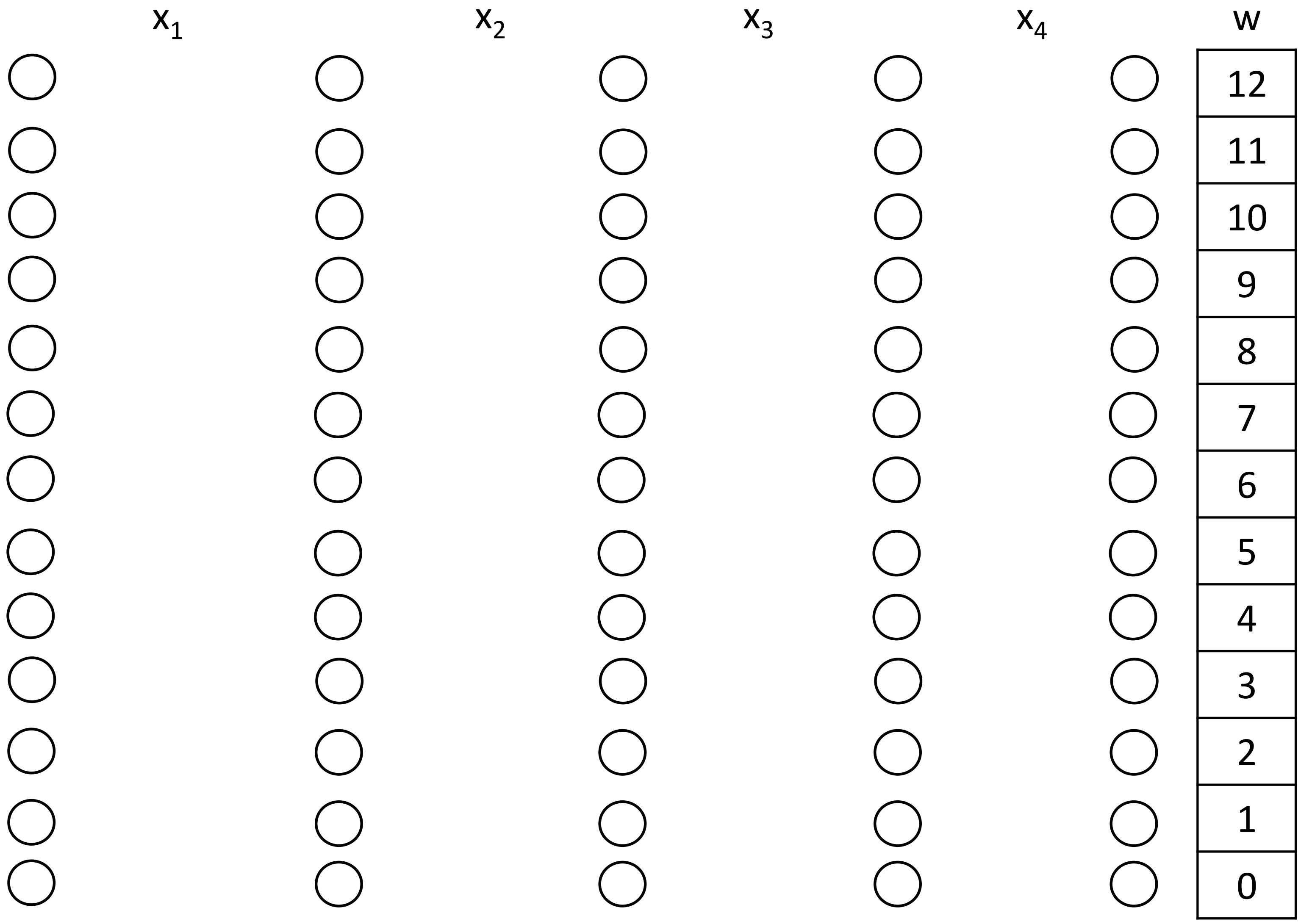
- Feasibility
  - Can we find a solution satisfying the constraint?
- Pruning
  - Can we eliminate values from the domains?

Binary knapsack

- The constraint $$\ell \leq \sum_{k \in R} w_k x_k \leq u$$ $$x_k \in \{0,1\}$$
- Feasibility
  - Use dynamic programming (pseudo-polynomial)
- Pruning
  - Exploit the dynamic programming table to prune the search
  - Forward phase (keep dependency links)
  - Backward phase (update dependency links to keep only feasible values)
  - Combine feasibility with pruning

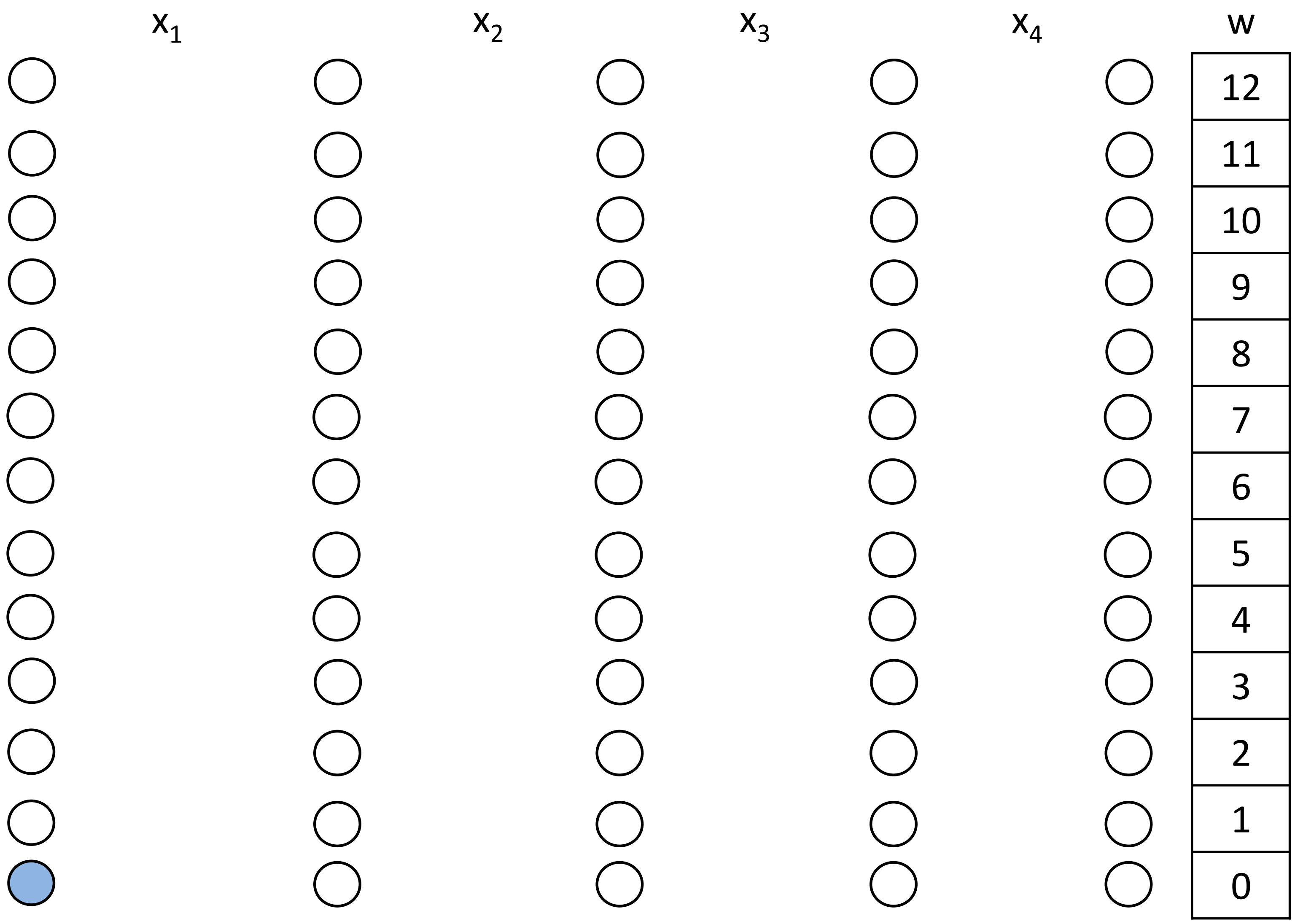
Binary knapsack: forward phase

- The constraint $10 \leq 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 12$



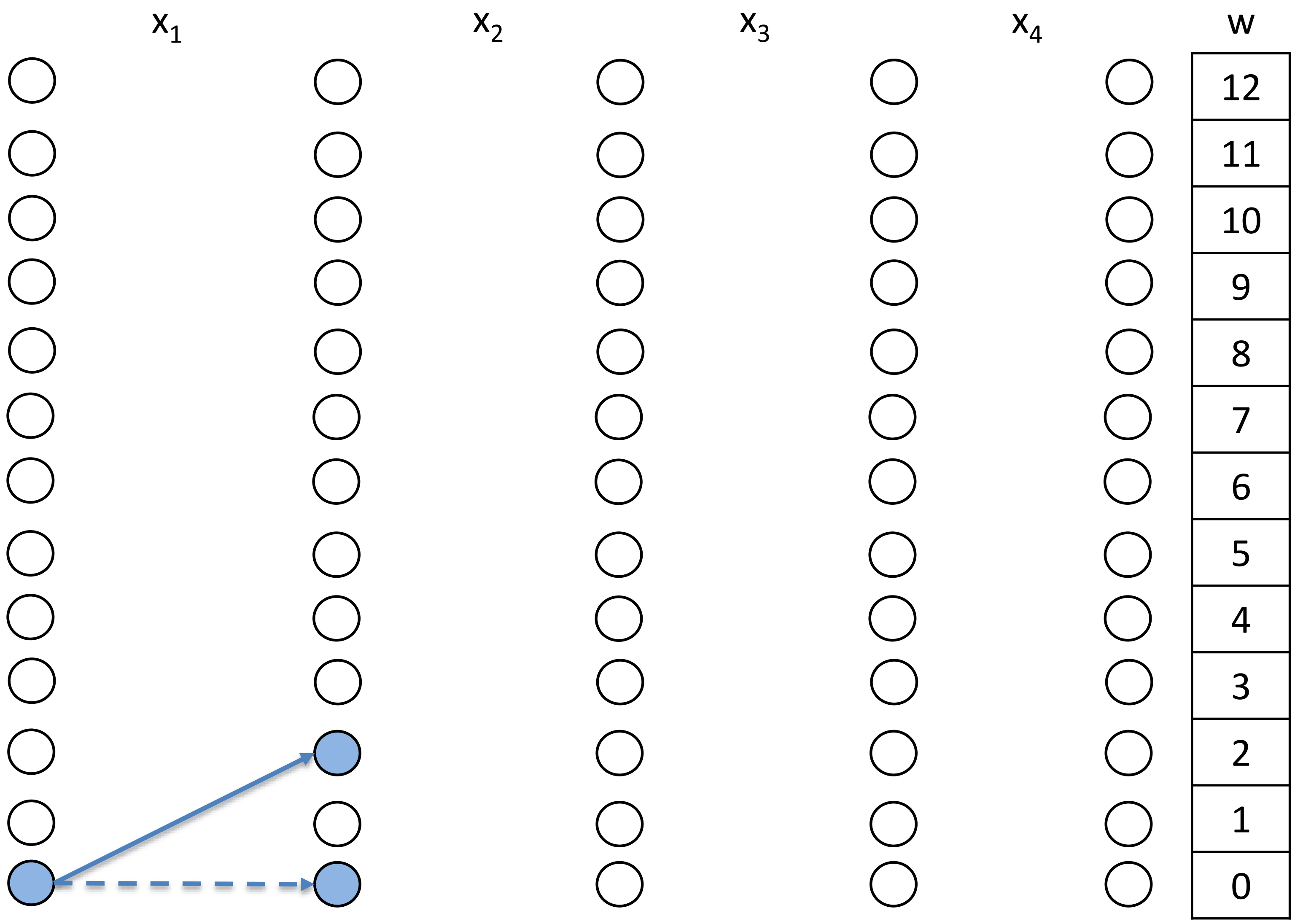
Binary knapsack: forward phase

- The constraint $10 \leq 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 12$



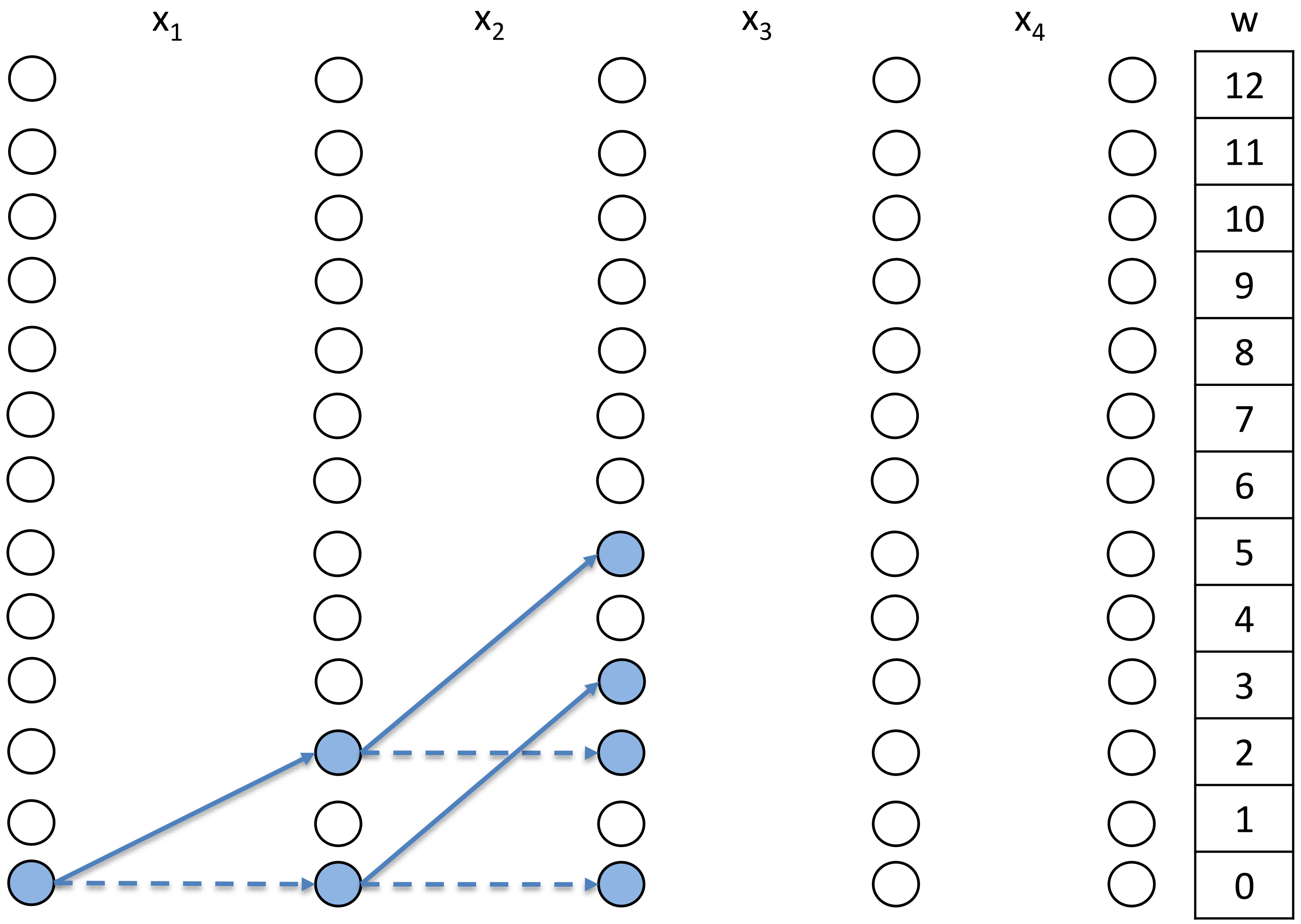
Binary knapsack: forward phase

- The constraint $10 \leq 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 12$



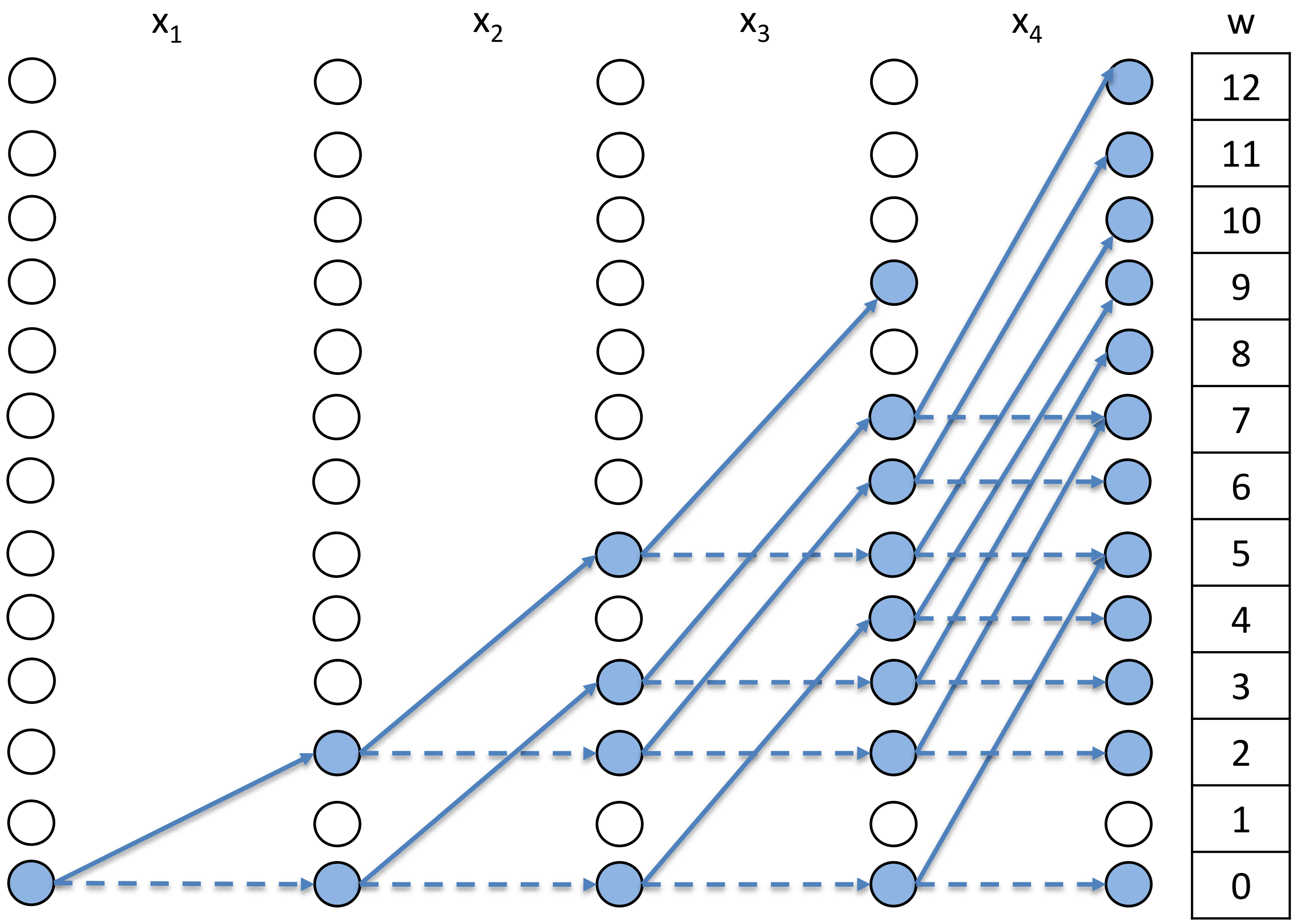
Binary knapsack: forward phase

- The constraint $10 \leq 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 12$



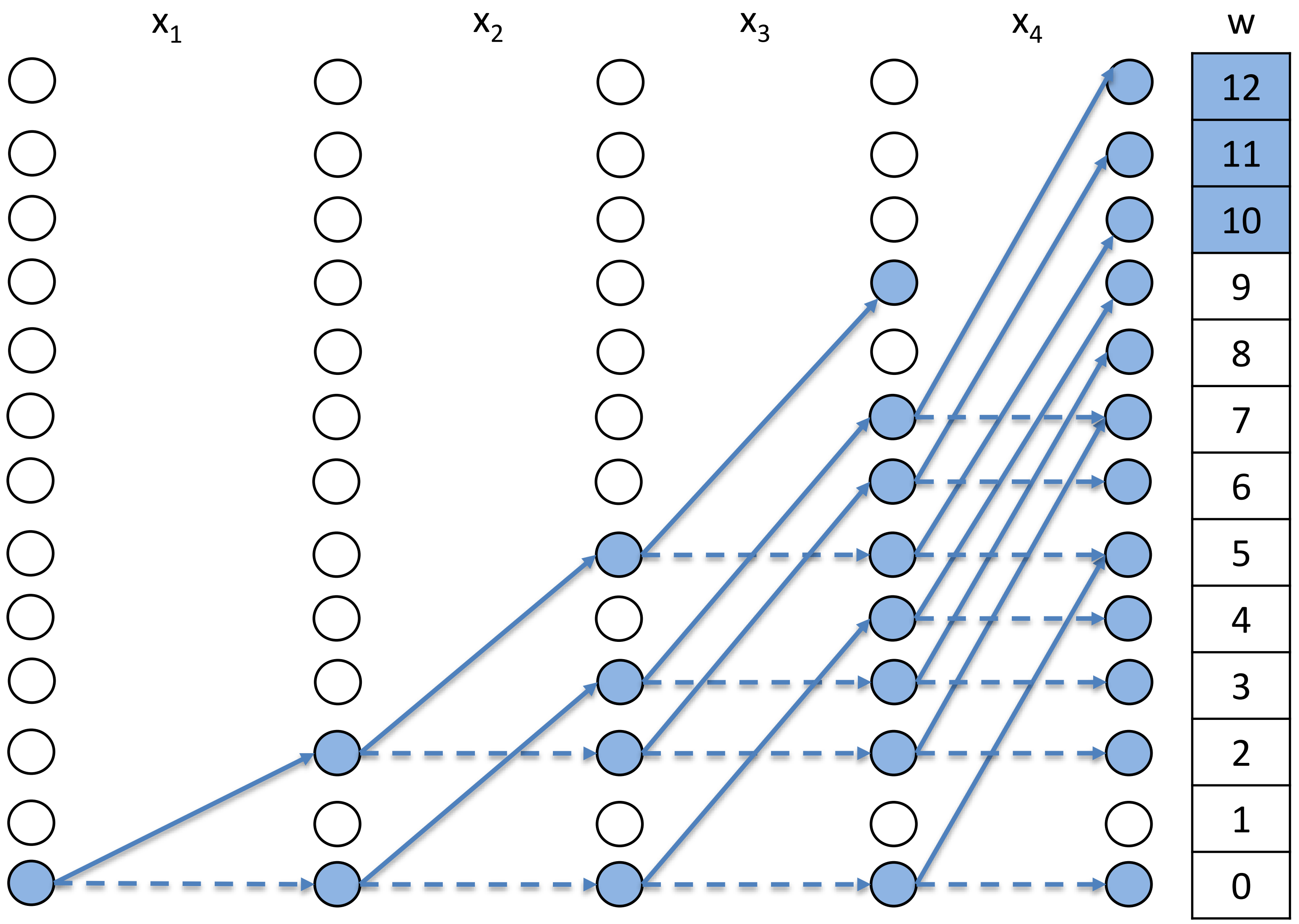
Binary knapsack: forward phase

- The constraint $10 \leq 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 12$



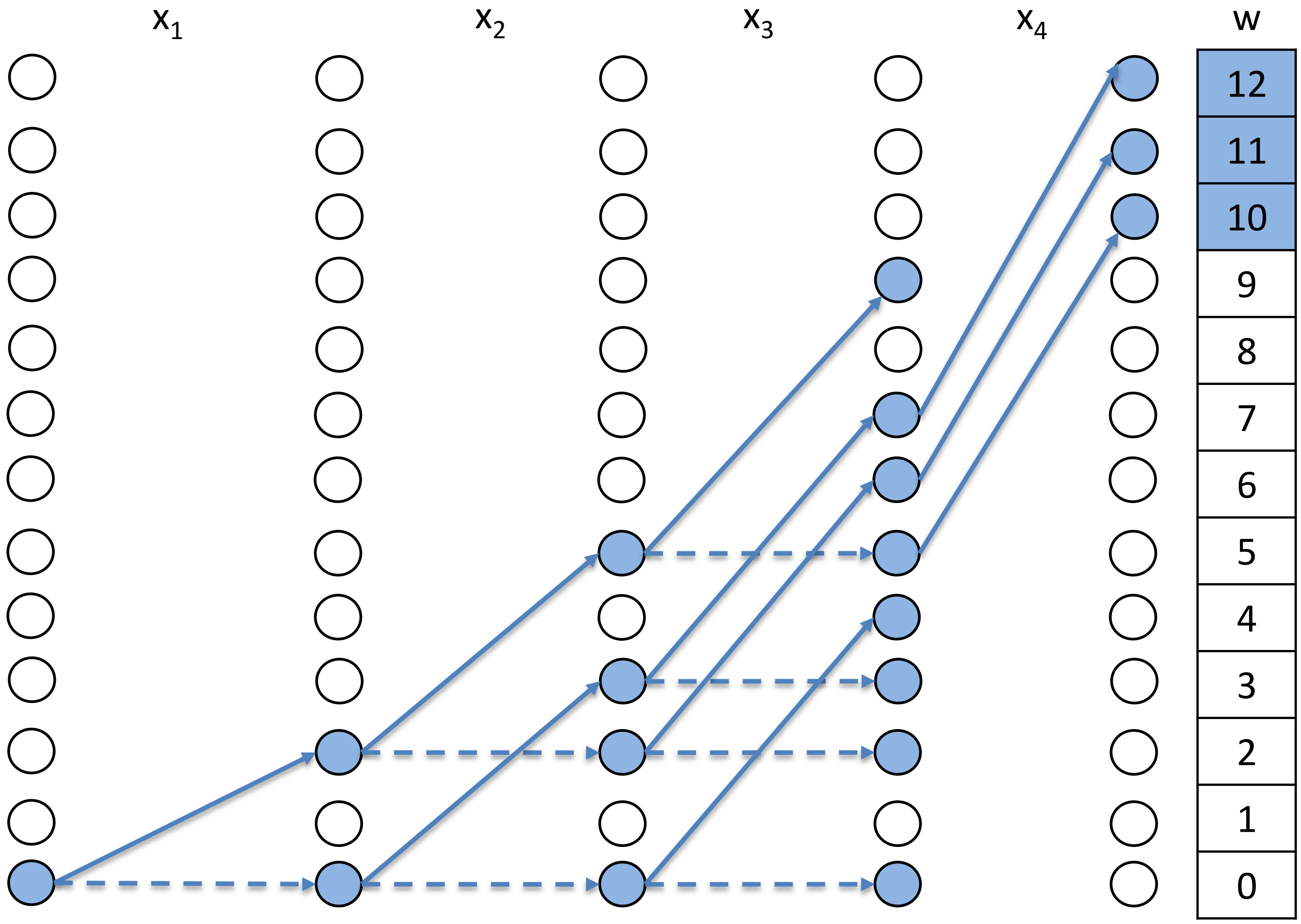
Binary knapsack: backward phase

- The constraint $10 \leq 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 12$



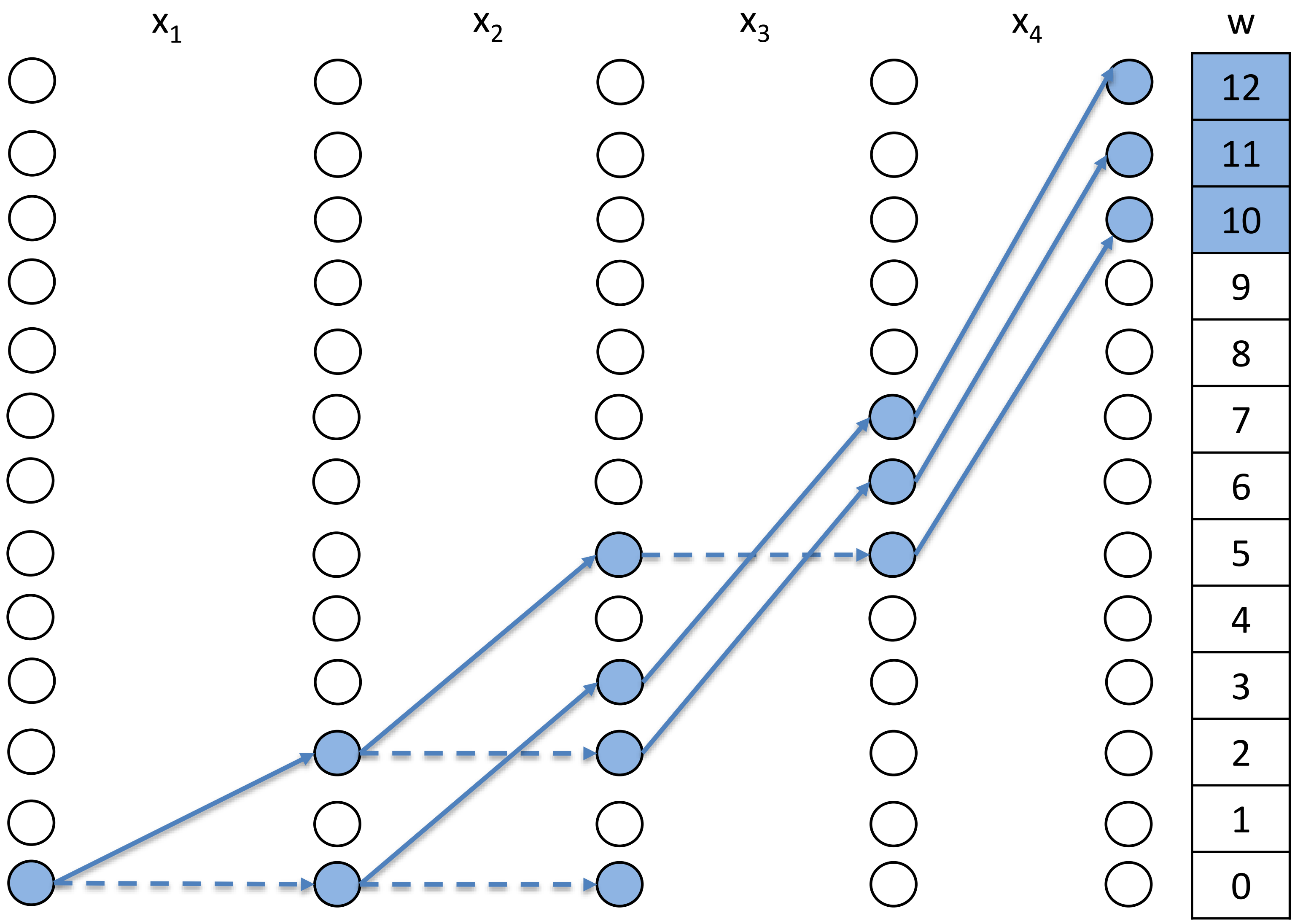
Binary knapsack: backward phase

- The constraint $10 \leq 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 12$



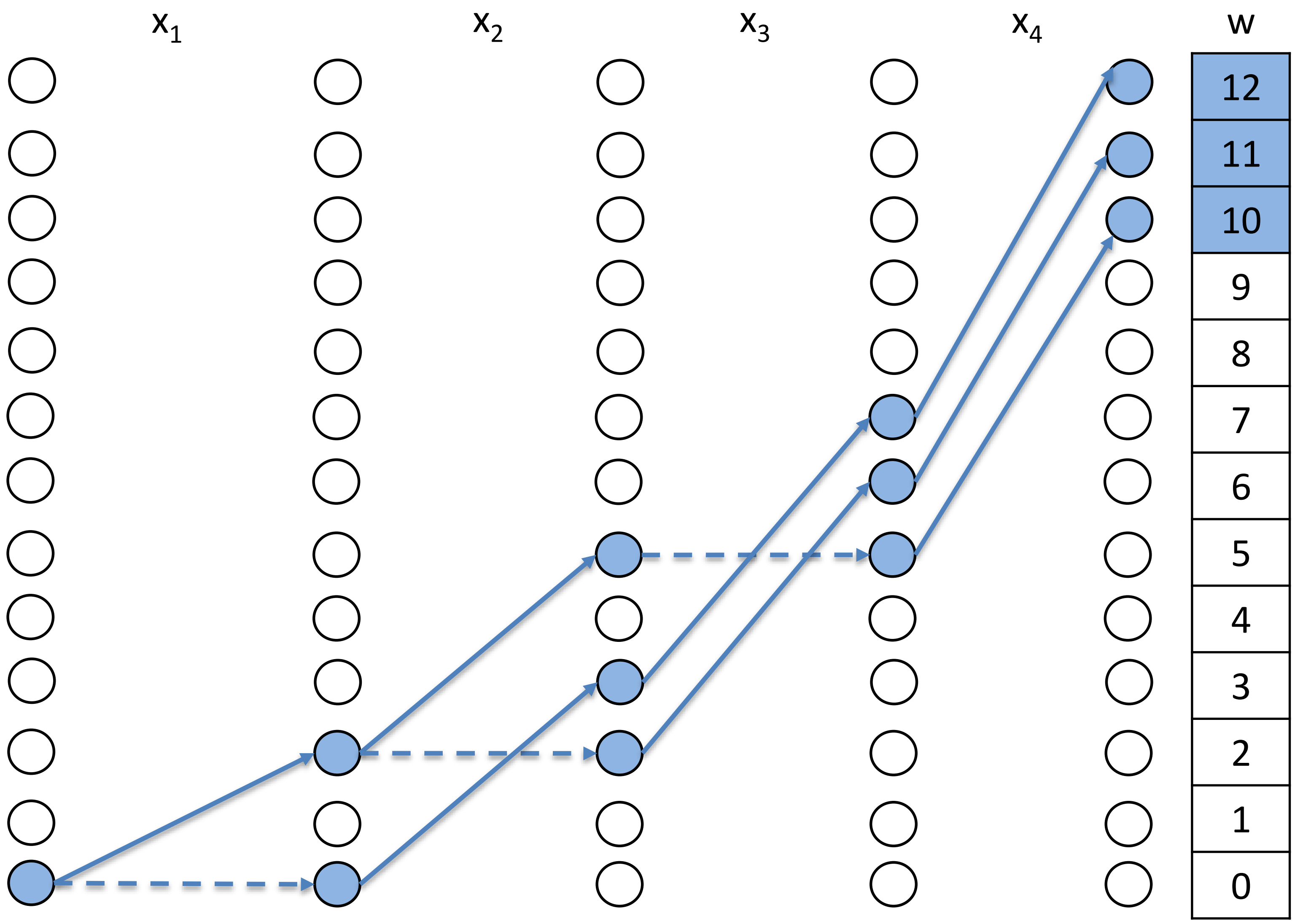
Binary knapsack: backward phase

- The constraint $10 \leq 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 12$



Binary knapsack: backward phase

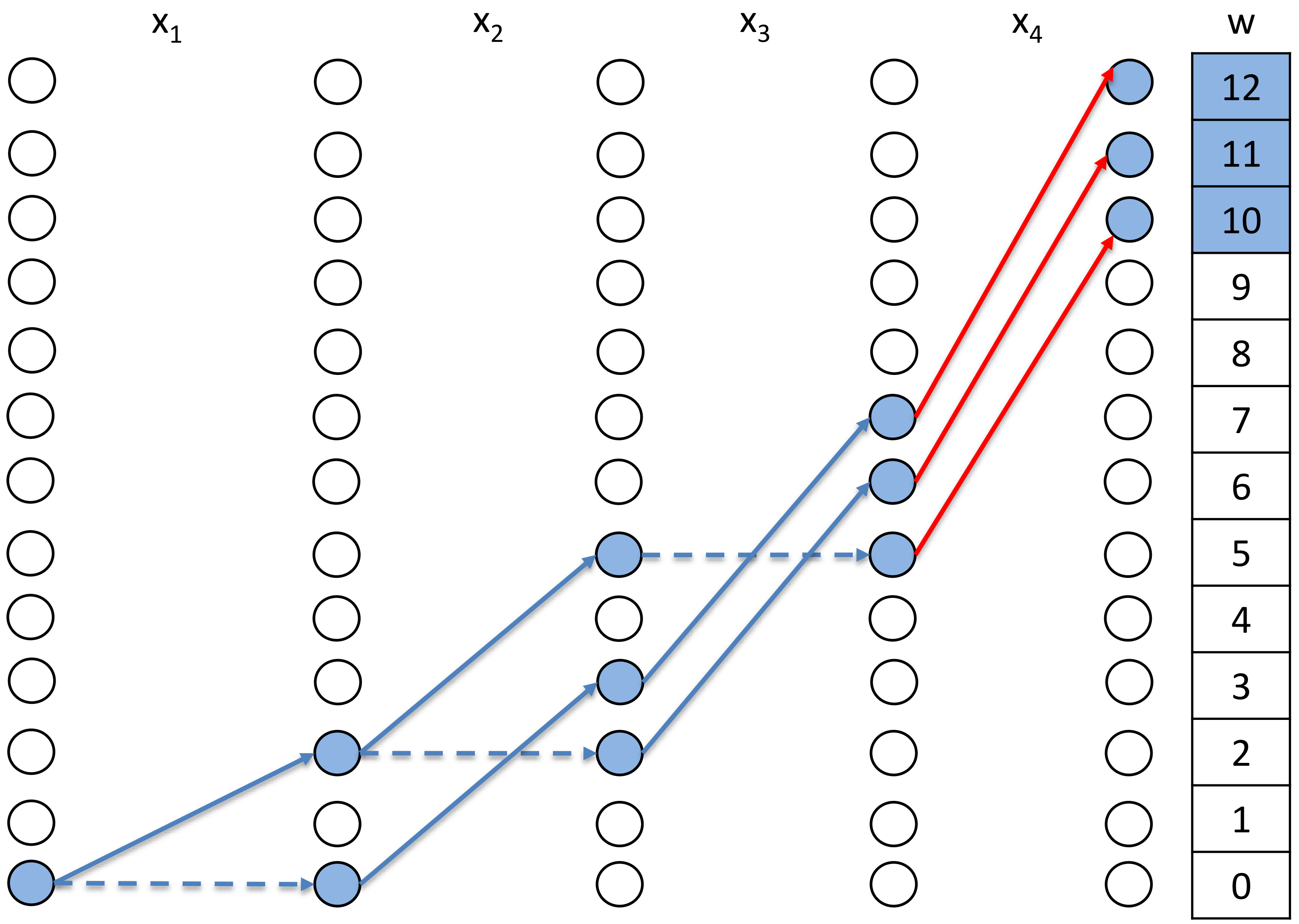
- The constraint $10 \leq 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 12$



Binary knapsack: backward phase

- The constraint $10 \leq 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 12$

x_4=1



Alldifferent constraint

- The constraint
  - Alldifferent(x_1, …, x_n)
- Feasibility
  - can we find values in the domains of the variables so that each two variables are assigned a different value?
- Pruning
  - are there values in the domain of a variable that the variable cannot take, i.e., if the variable takes that value, then there is no solution.

Alldifferent representation

$x_1 \in \{1, 2\}$

$x_2 \in \{2, 3\}$

$x_3 \in \{1, 3\}$

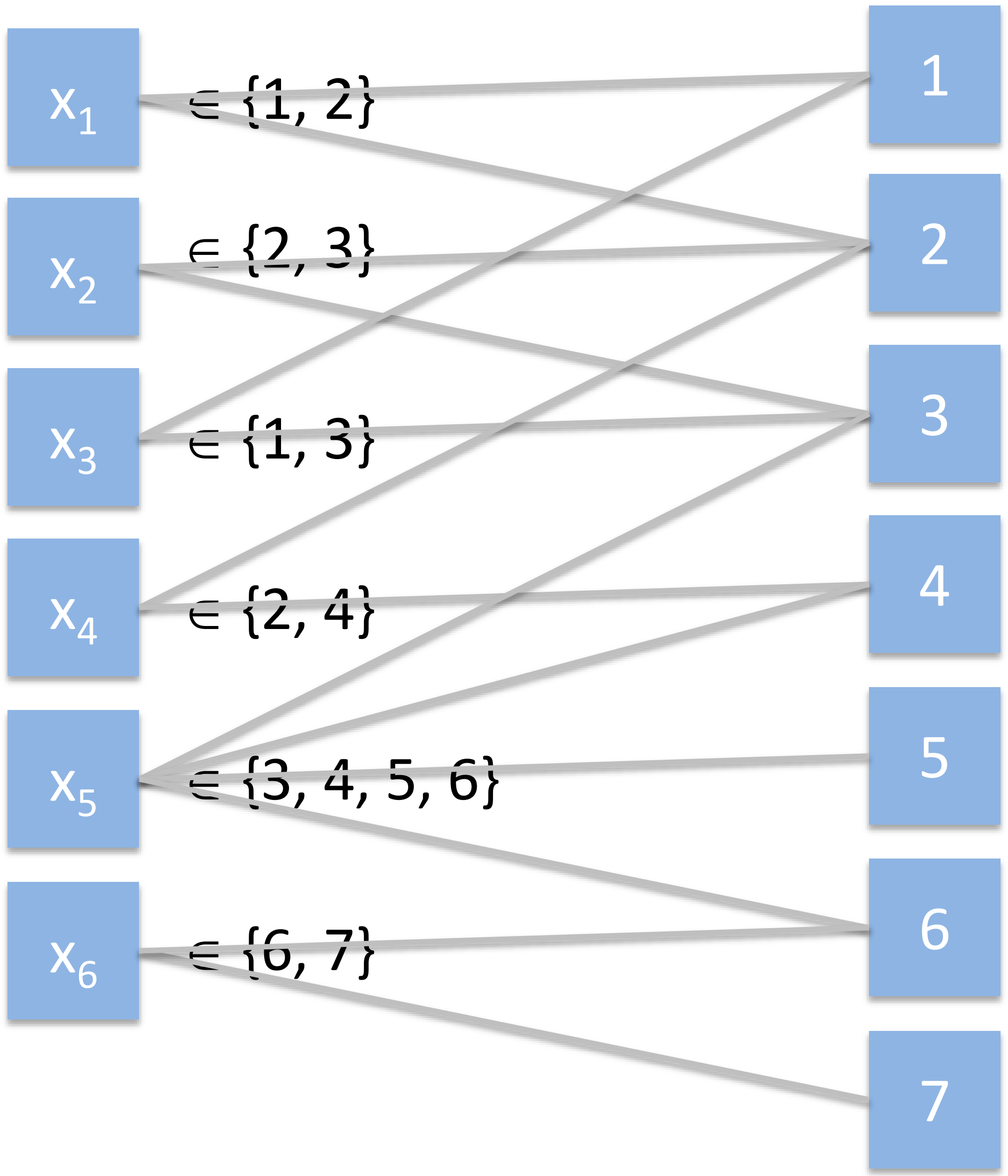
$x_4 \in \{2, 4\}$

$x_5 \in \{3, 4, 5, 6\}$

$x_6 \in \{6, 7\}$

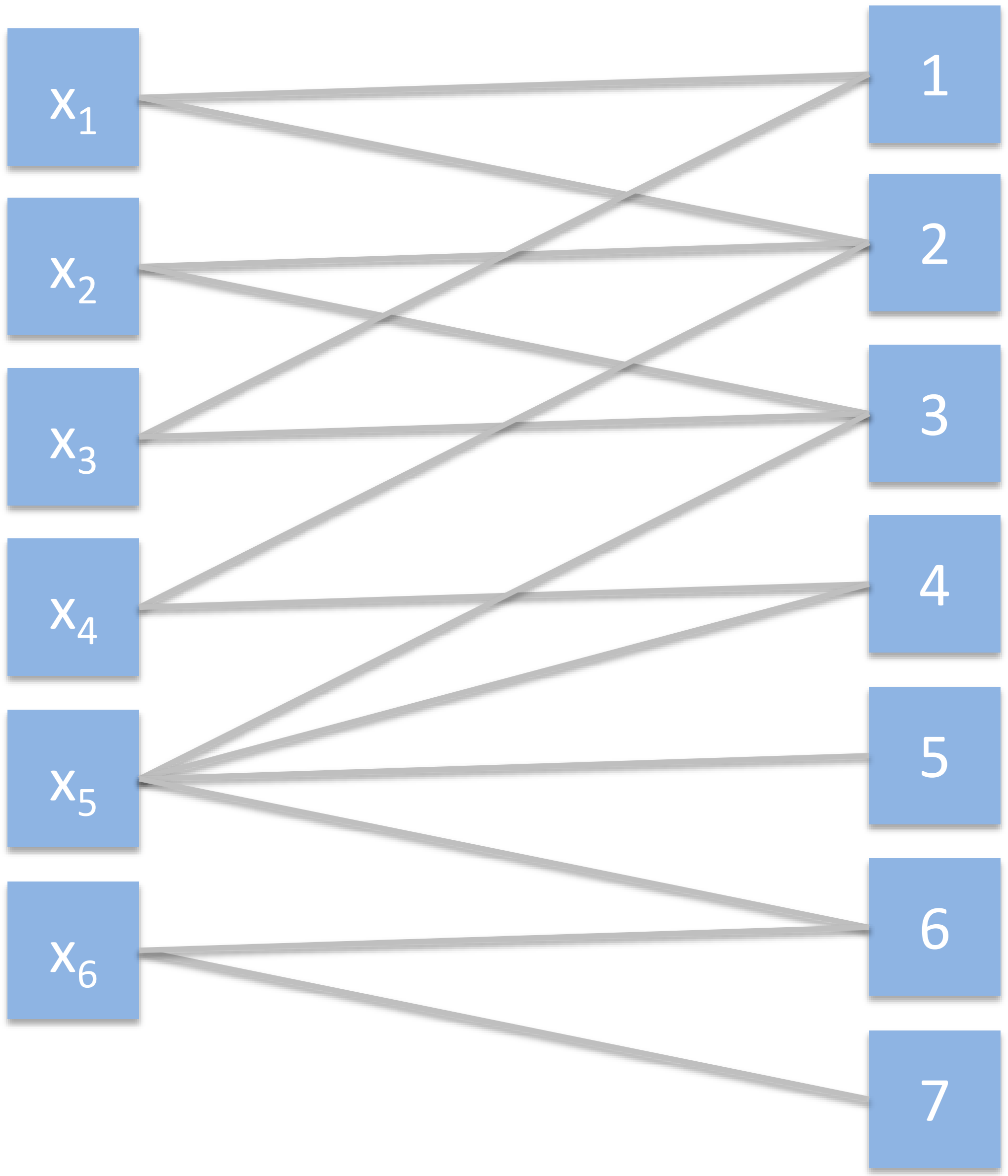
If all the variables take different values can x_4 take the value 2?

Alldifferent representation

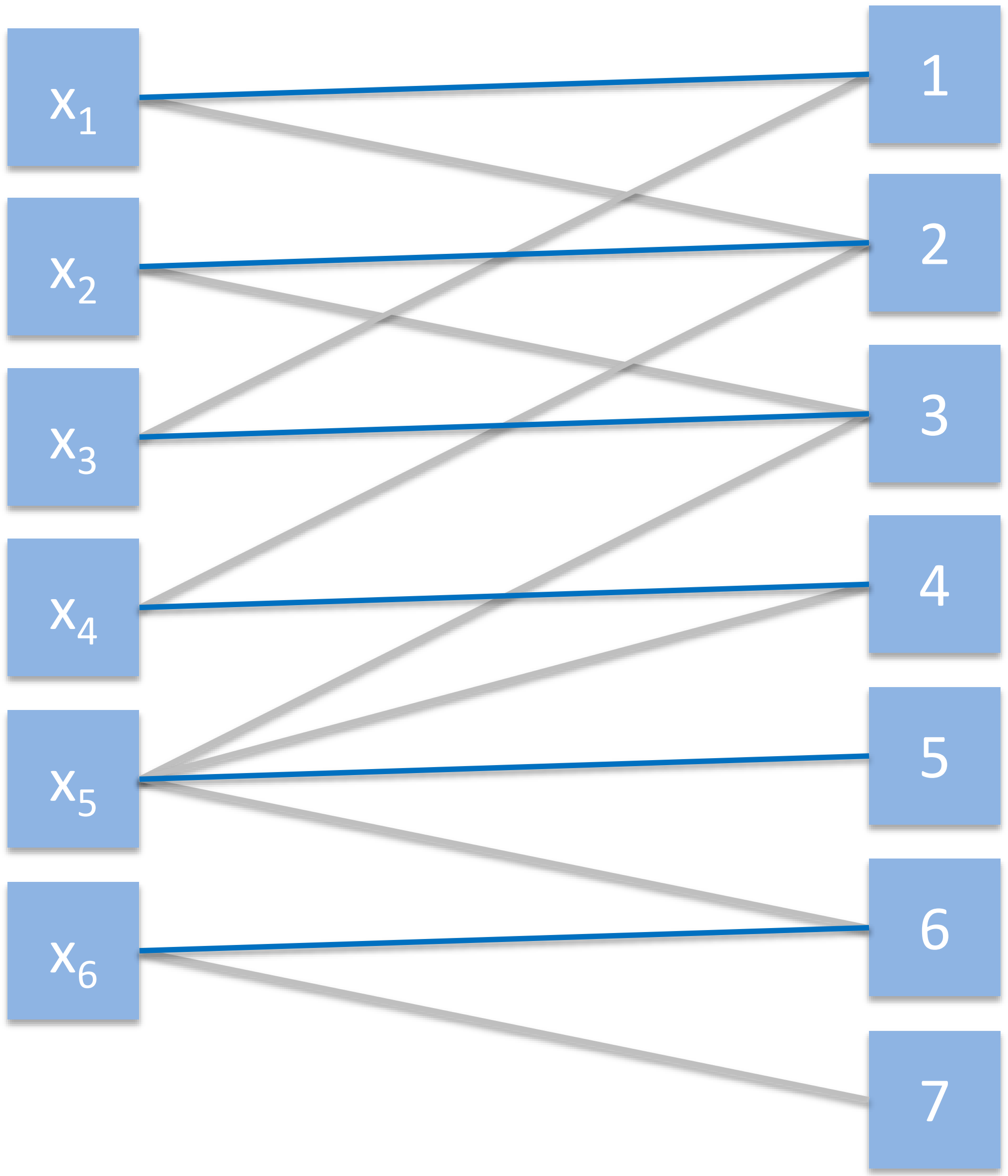


Can x_4 take the value 2?

Alldifferent feasibility



Alldifferent feasibility



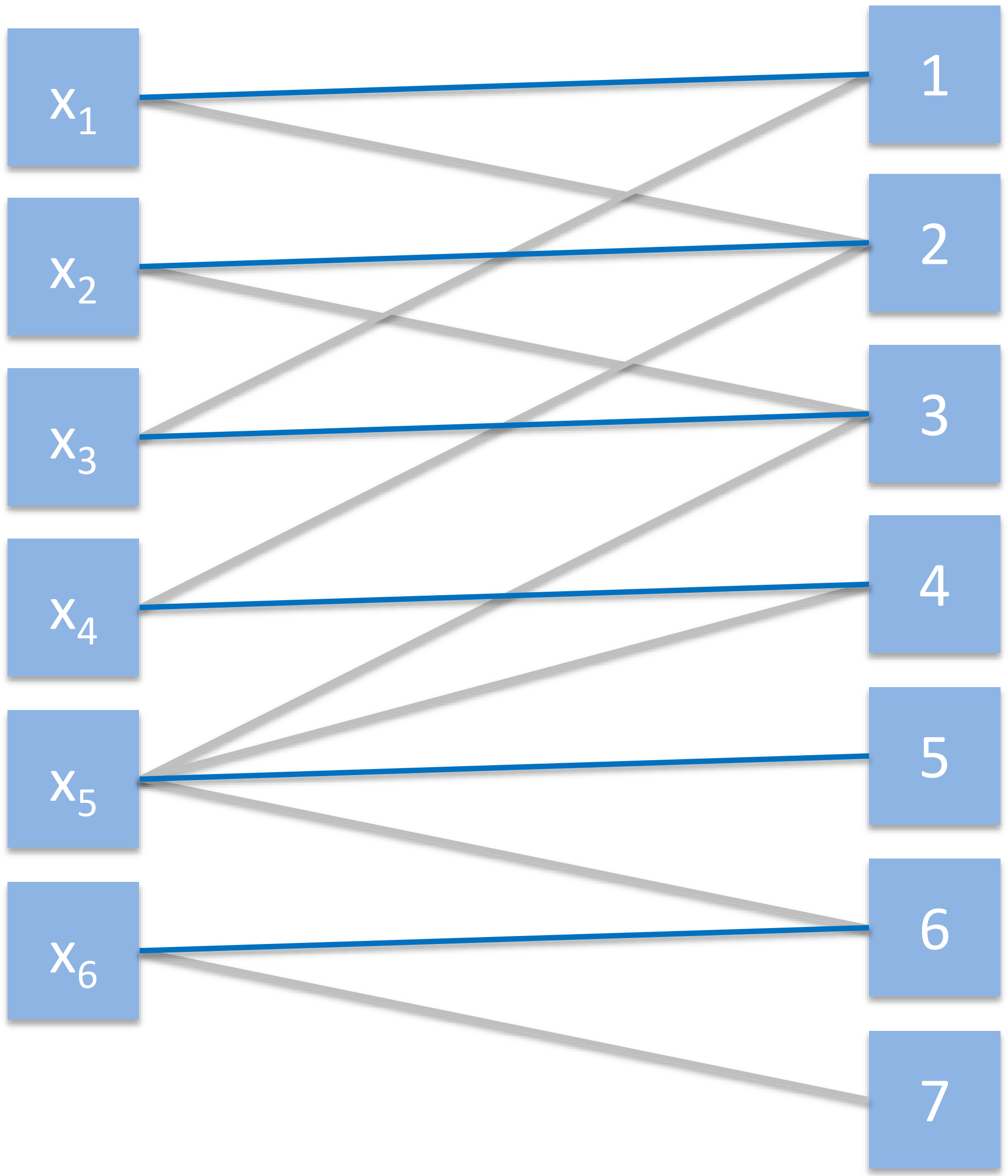
Created a bipartite graph

- nodes for variables
- nodes for values
- edges between variables and values

Alldifferent and matching

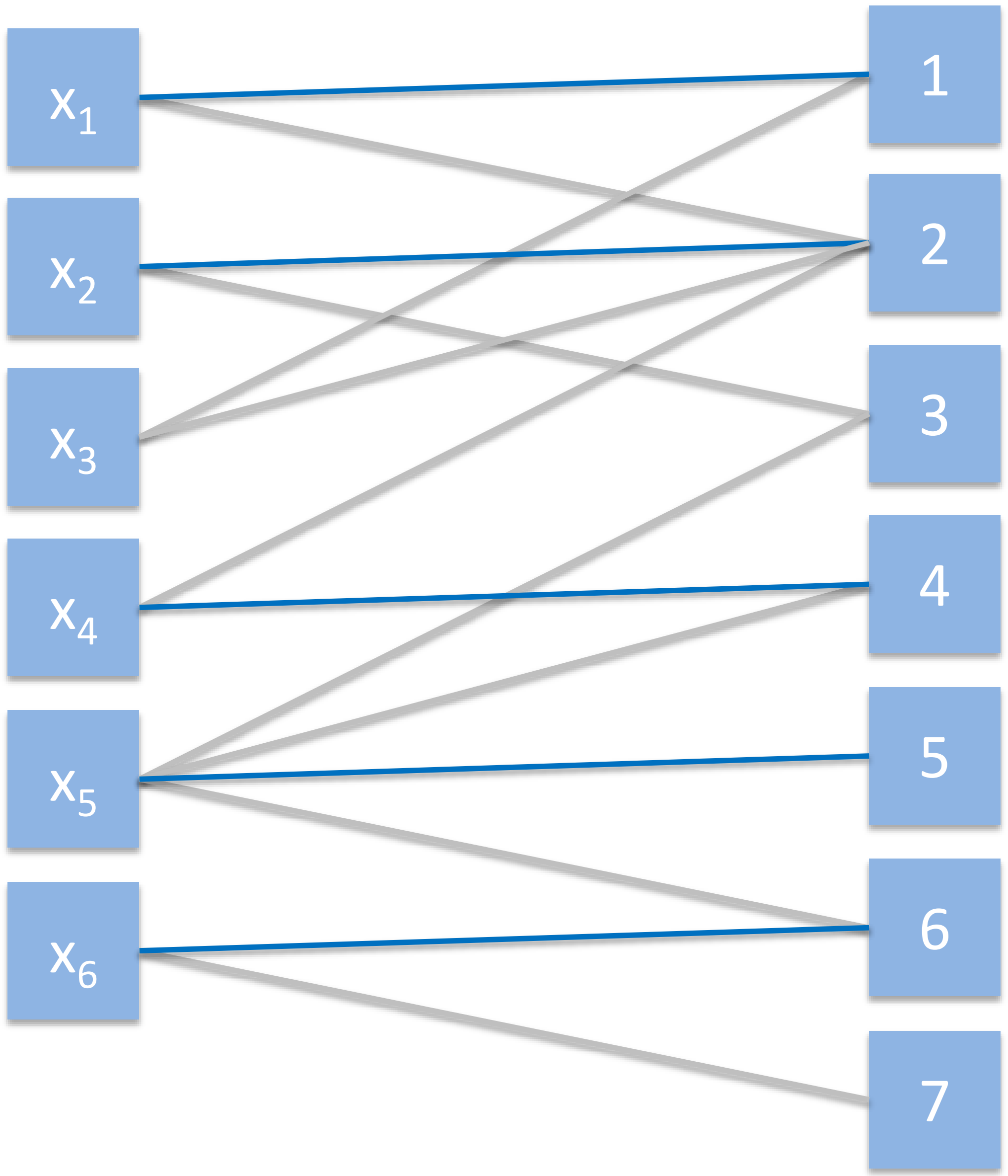
- A matching for a graph G=(V,E) is a set of edges in E such that no two edges in E share a vertex.
- A maximum matching M for a graph G is a matching with the largest number of edges.
- Feasibility
    - finding a maximum matching in a bipartite graph.
    - if the maximum matching has a size equal to the number of variables, then the constraint is feasible; otherwise, it is not feasible

Alldifferent feasibility



Feasible constraint

Alldifferent feasibility



Not feasible constraint

Alldifferent and matching

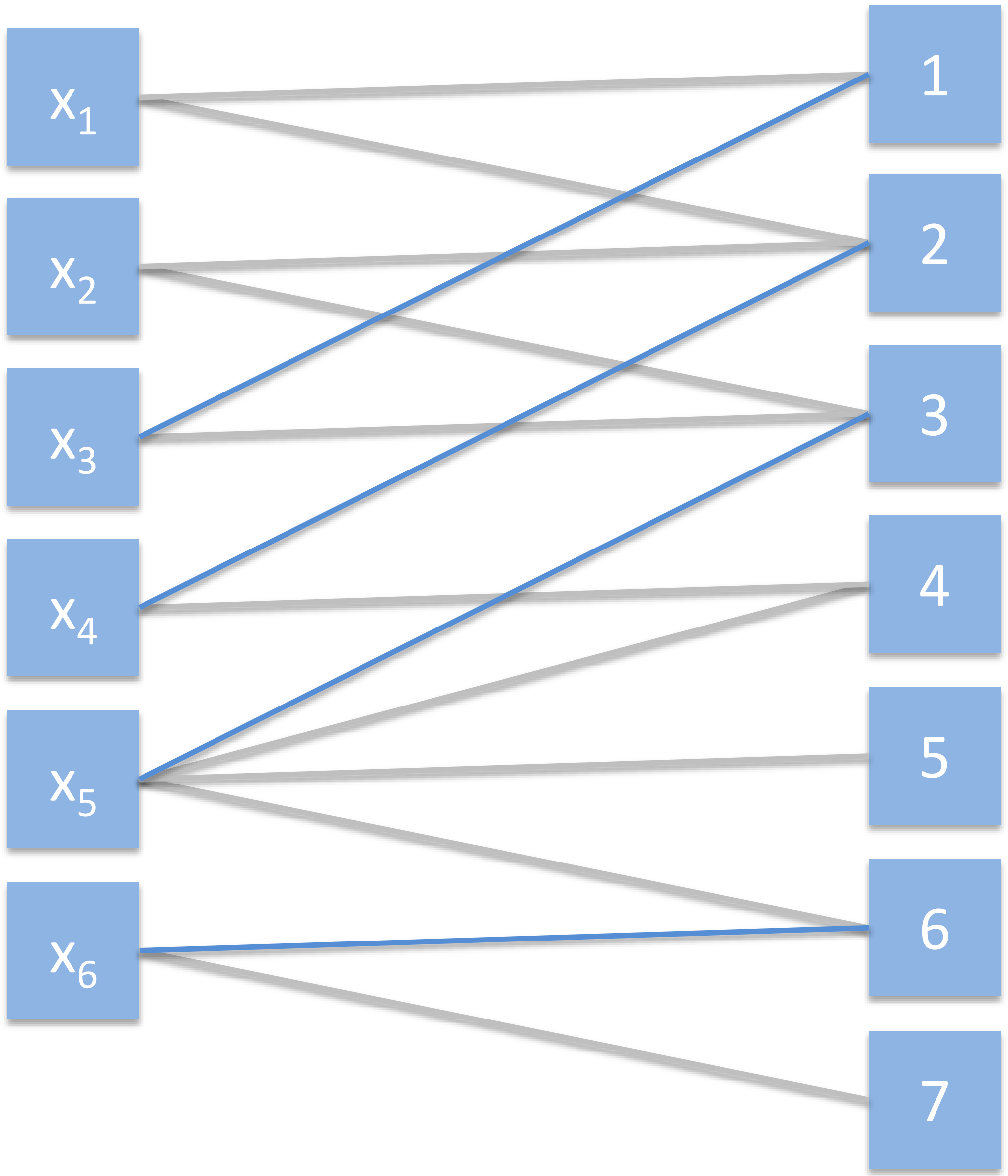
- How to find a maximum matching?
  - Start with any matching
  - Improve the matching
- When no improvement is possible
  - We have a maximum matching

Maximum matching

- How to find a maximum matching?
  - Start with any matching
  - Improve the matching
- How to find an improvement?
  1. Start from a free vertex x
  2. If there us an edge (x,v) where v is not matched, then insert (x,v) in the matching
  3. Otherwise, take a vertex v matched to y.

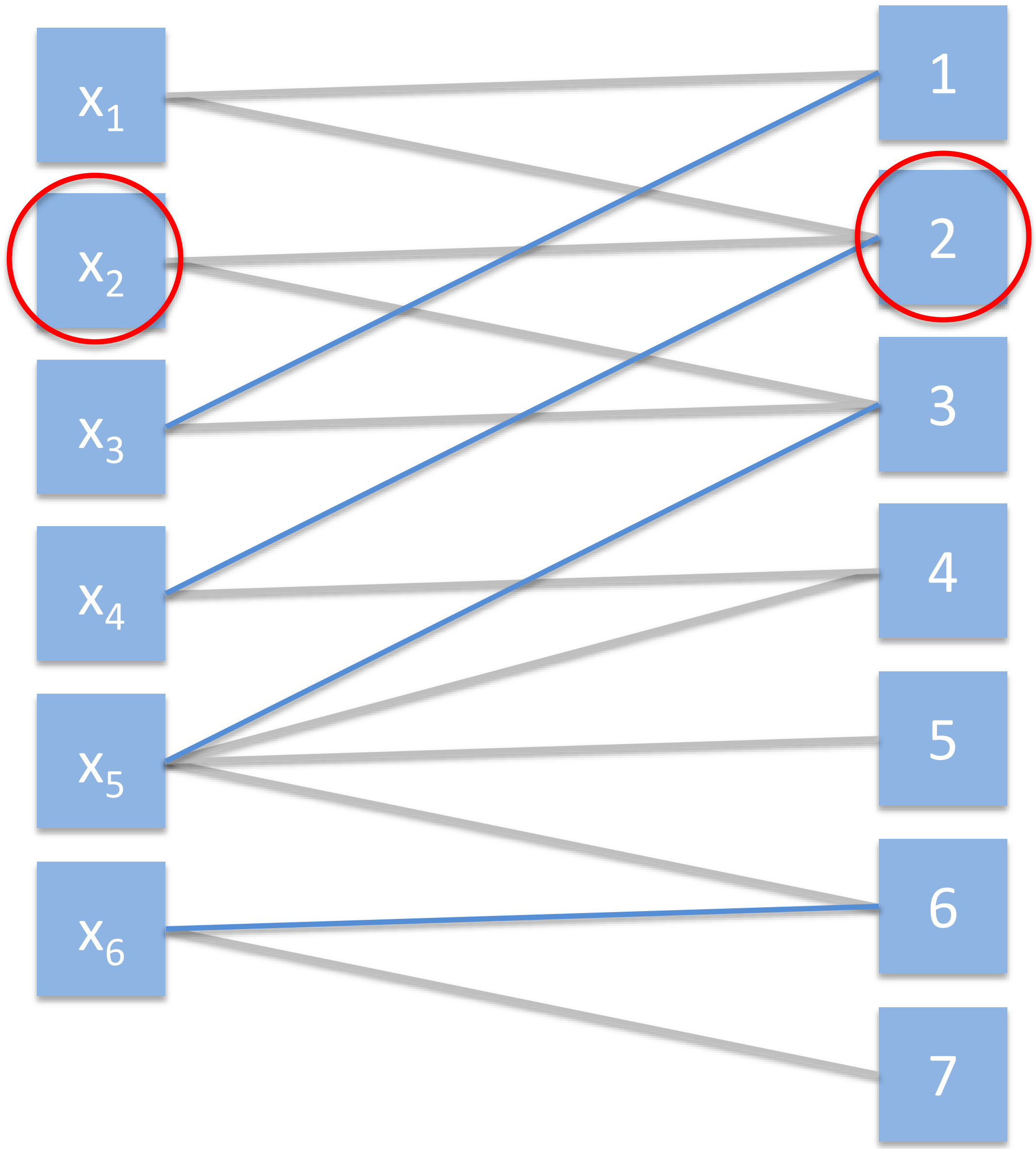
     remove (y, v) and add (x,v) from the matching and restart at step 2 with y instead of x

Maximum matching



Start with a matching

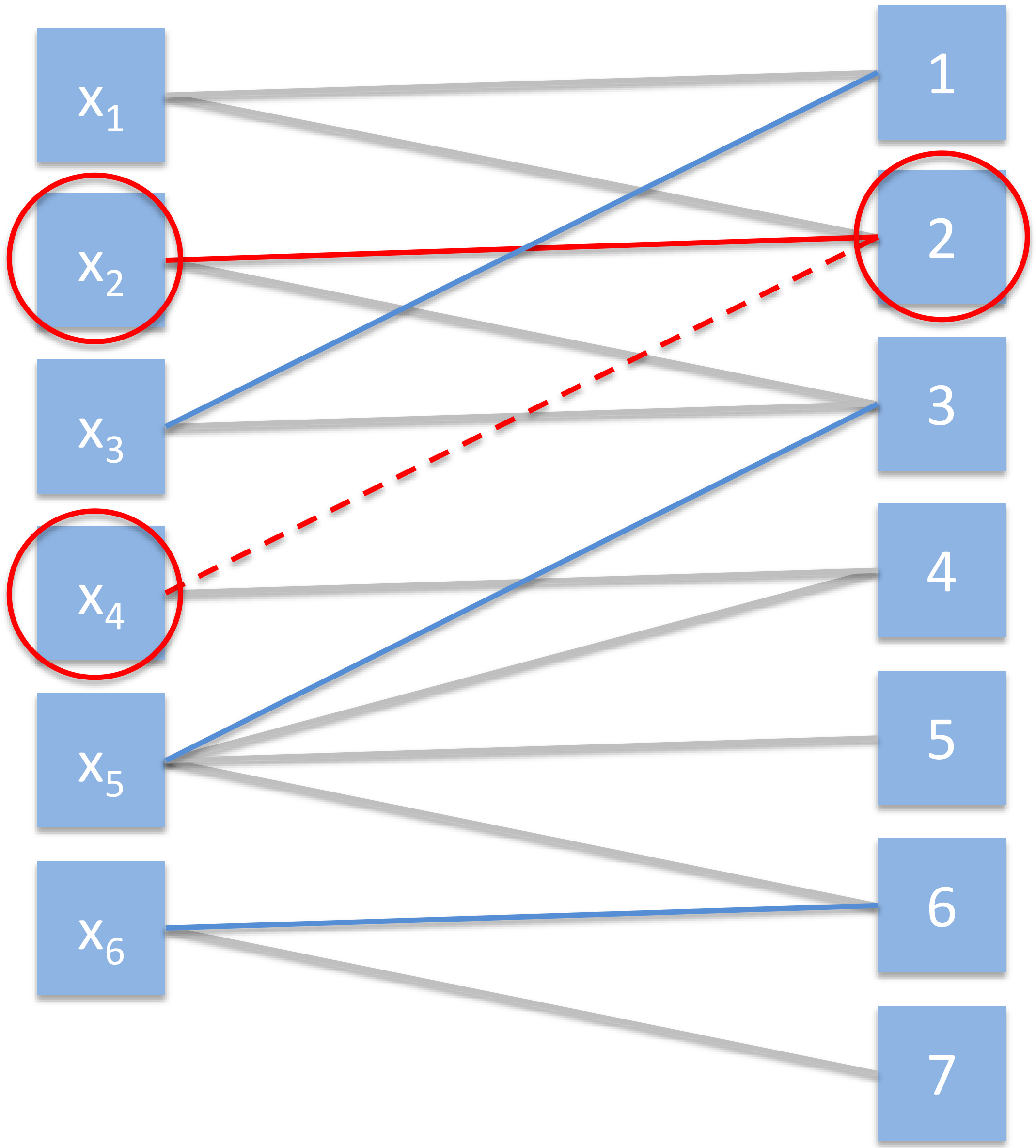
Maximum matching



Start with a matching

Select x_2 and 2

Maximum matching



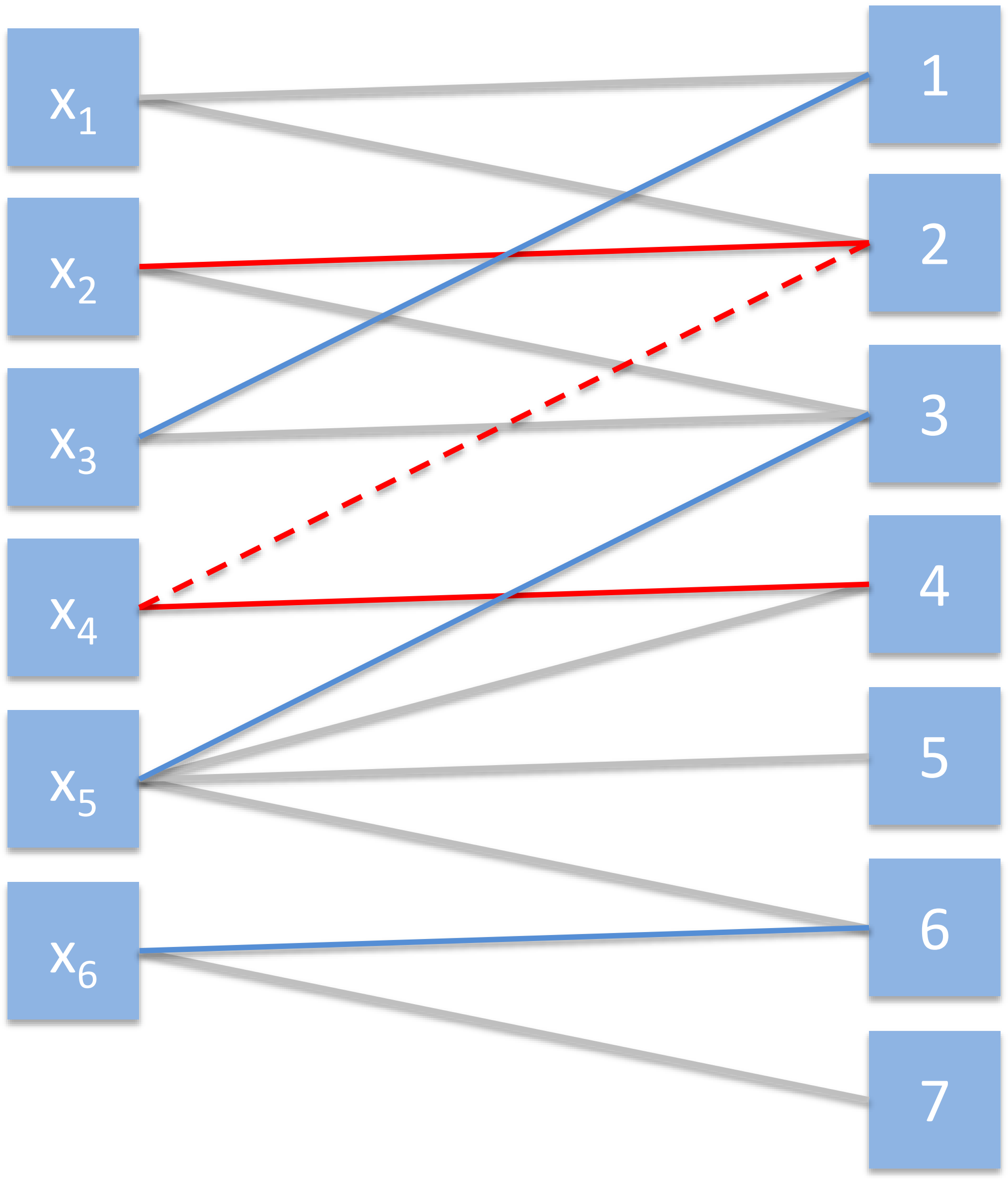
Start with a matching

Select x_2 and 2

Remove edge $x_4 – 2$

Start again with x_4

Maximum matching



Start with a matching

Select x_2 and 2

Remove edge $x_4 – 2$

Start again with x_4

Add x_4 - 4

Alternating path

- An alternating path P for a matching M is a path from a vertex x in X to a vertex v in V (both of which are free) such that the edges in the path are alternatively in E\M and M

- Alternating path has odd number of edges

- Alternating path improves a matching

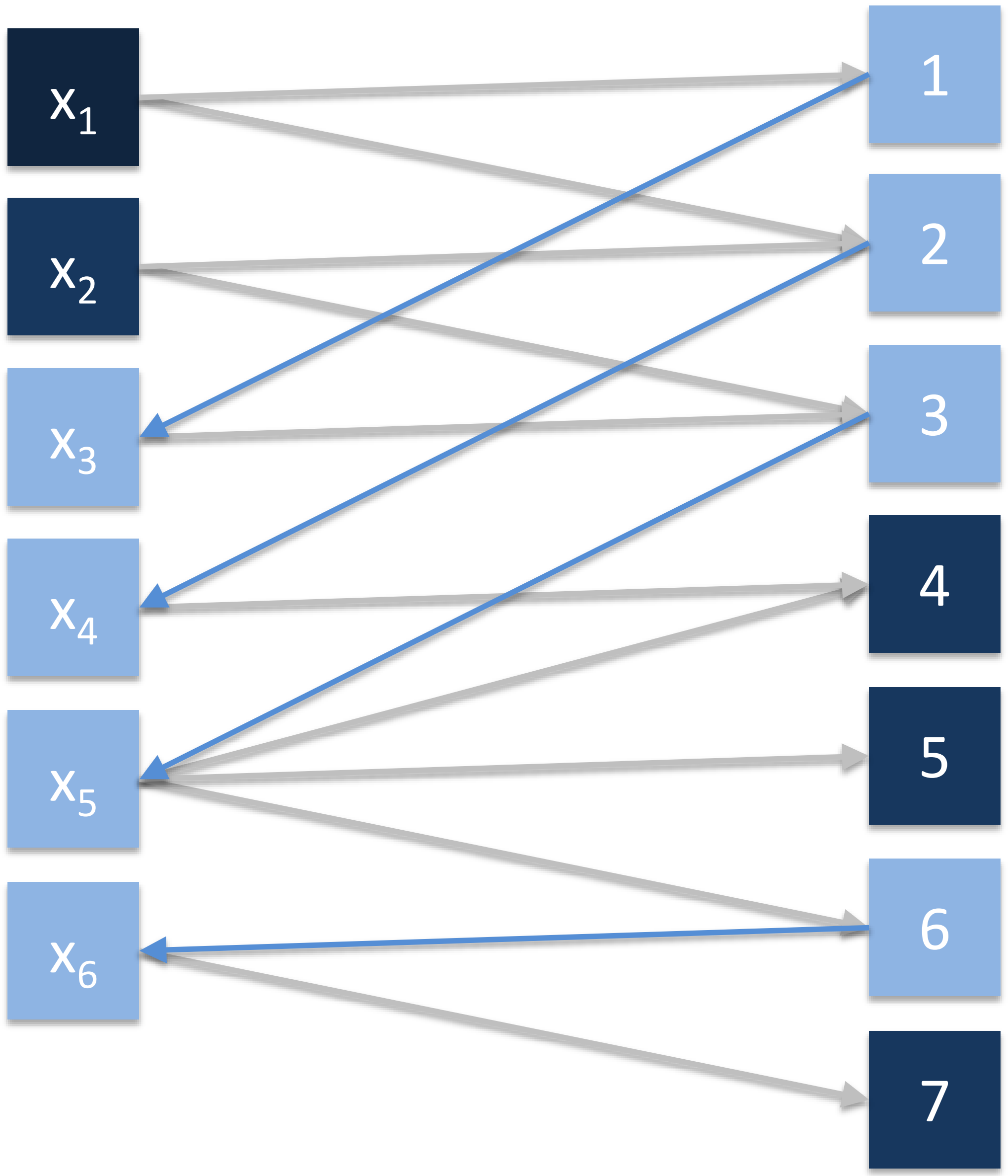
Finding an alternating path

- Create a directed graph
  - Edges in the matching are oriented from right to left
  - Edges not in the matching are oriented from left to right

- An alternating path is thus a path starting from a free vertex x and ending in another free vertex v

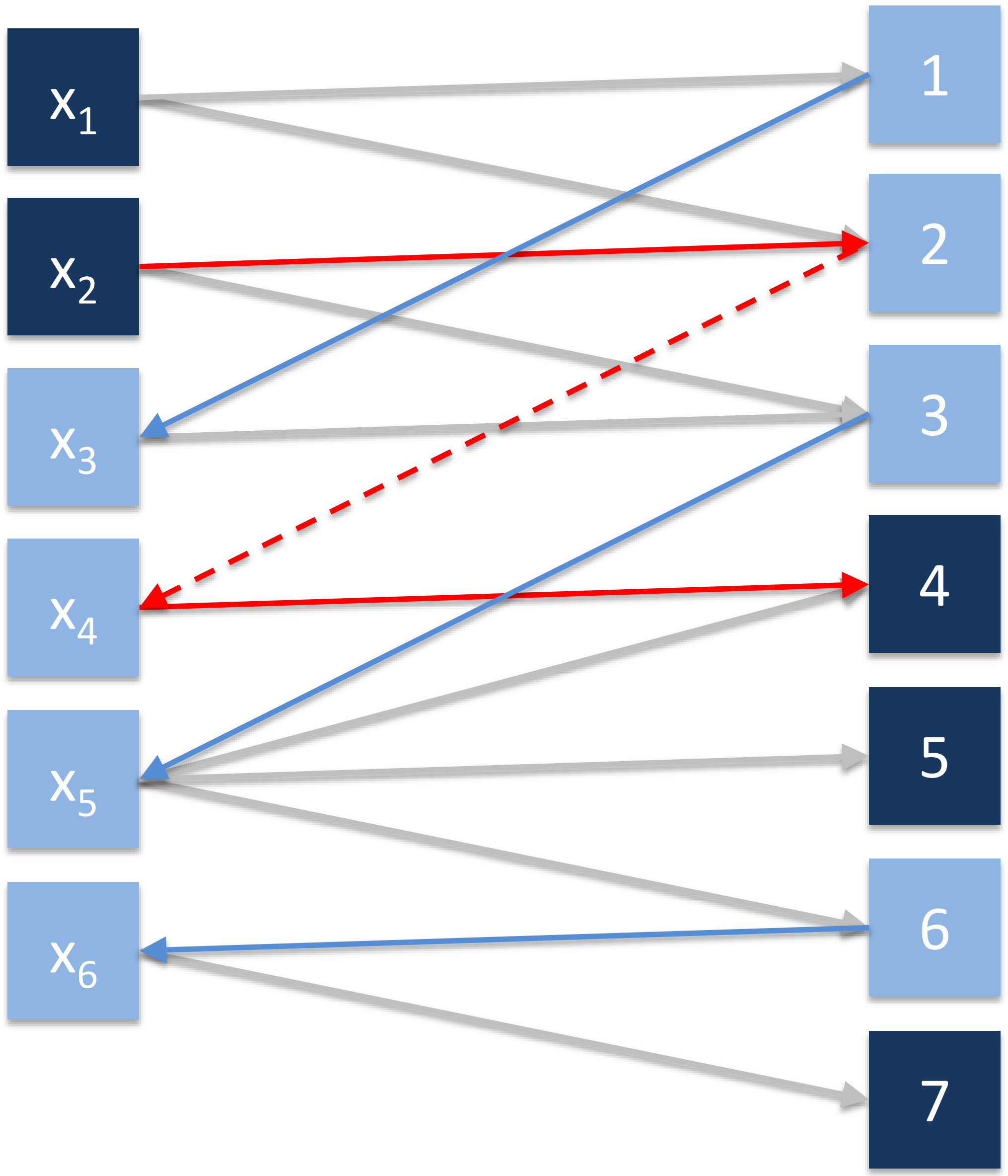
- Find such a path with Depth-First Search
  - Complexity $O(|V| + |E|)$ where V is the set of vertices and E is the set of edges

Maximum matching



Directed graph

Maximum matching

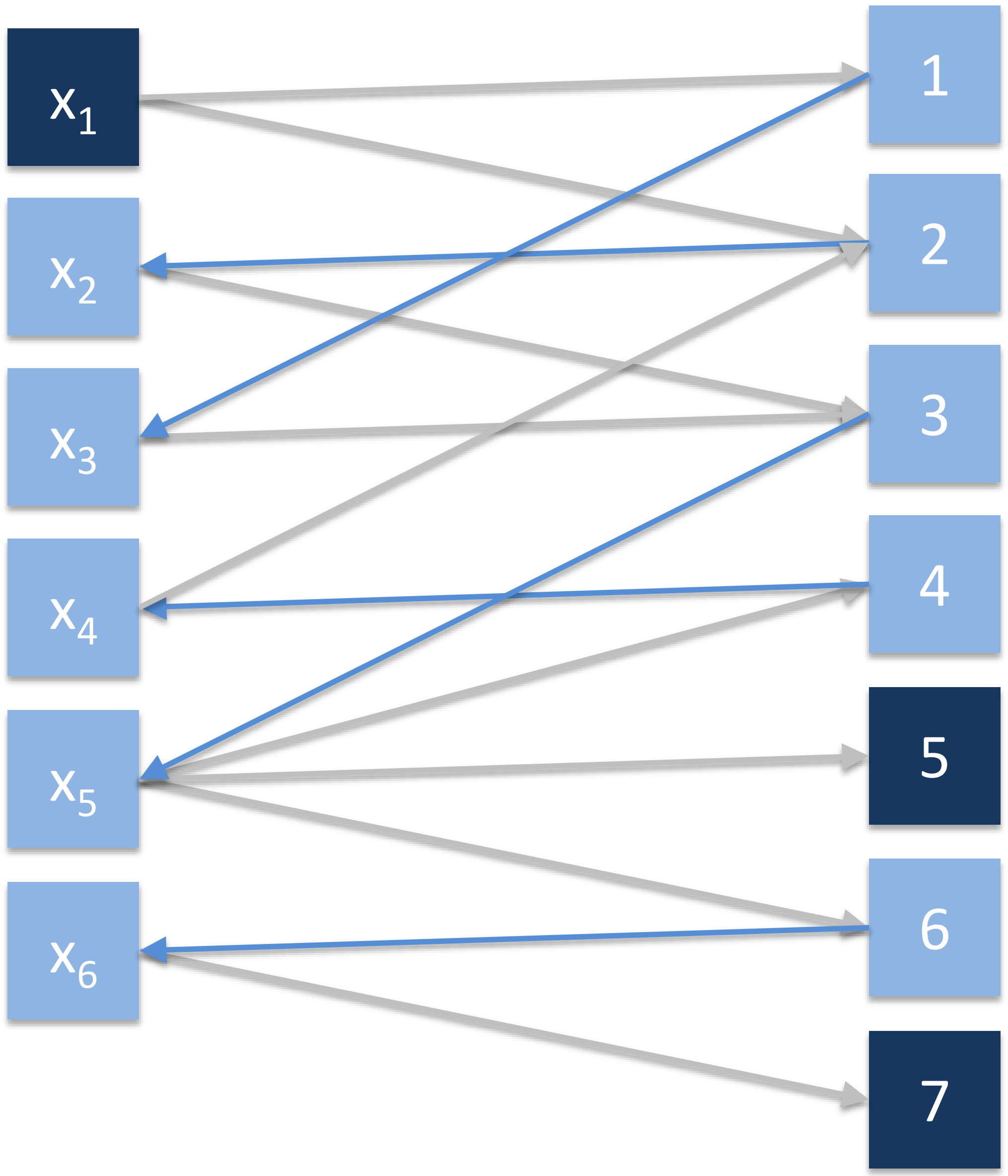


Directed graph

An alternating path

Change the direction

Maximum matching



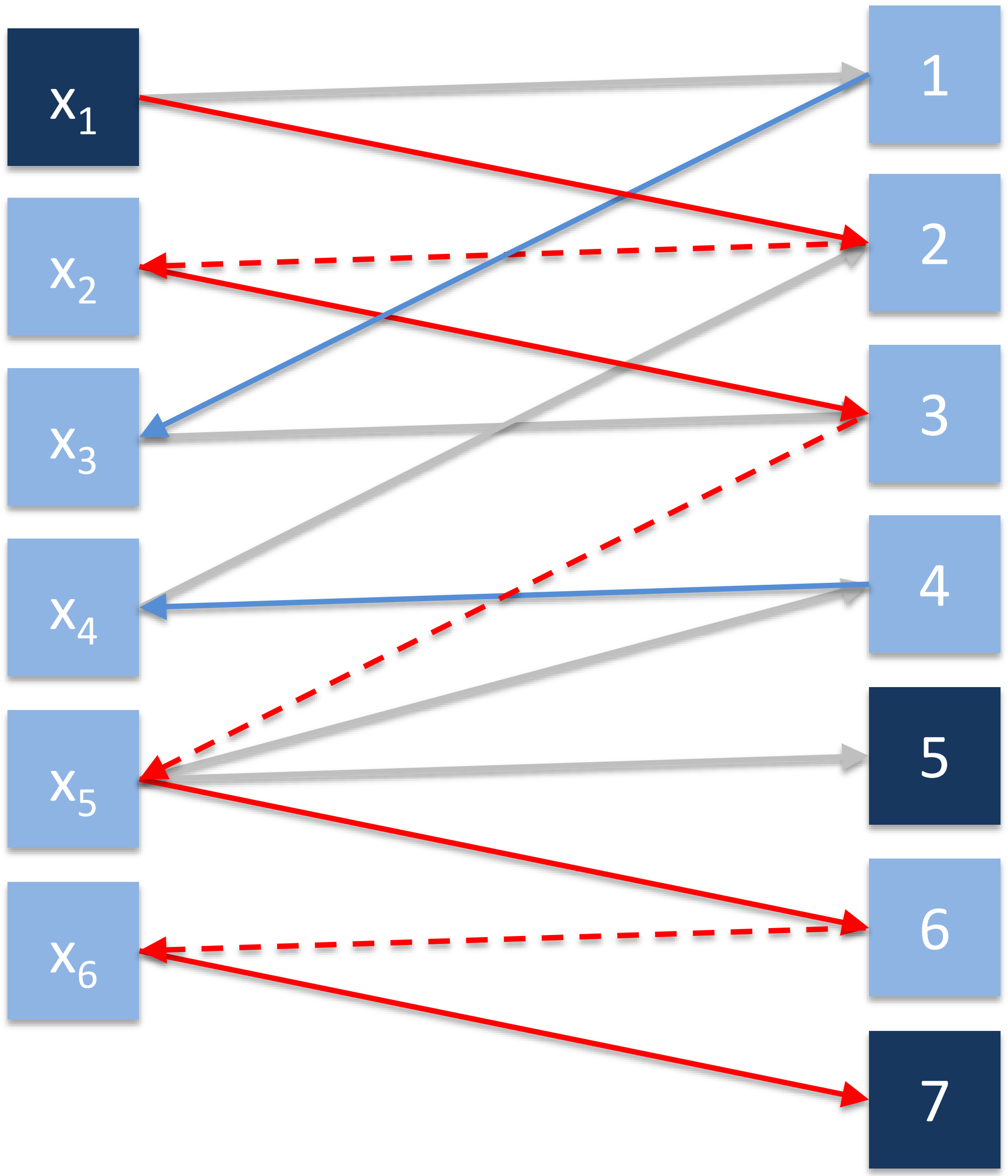
Directed graph

An alternating path

Change the direction

Repeat

Maximum matching



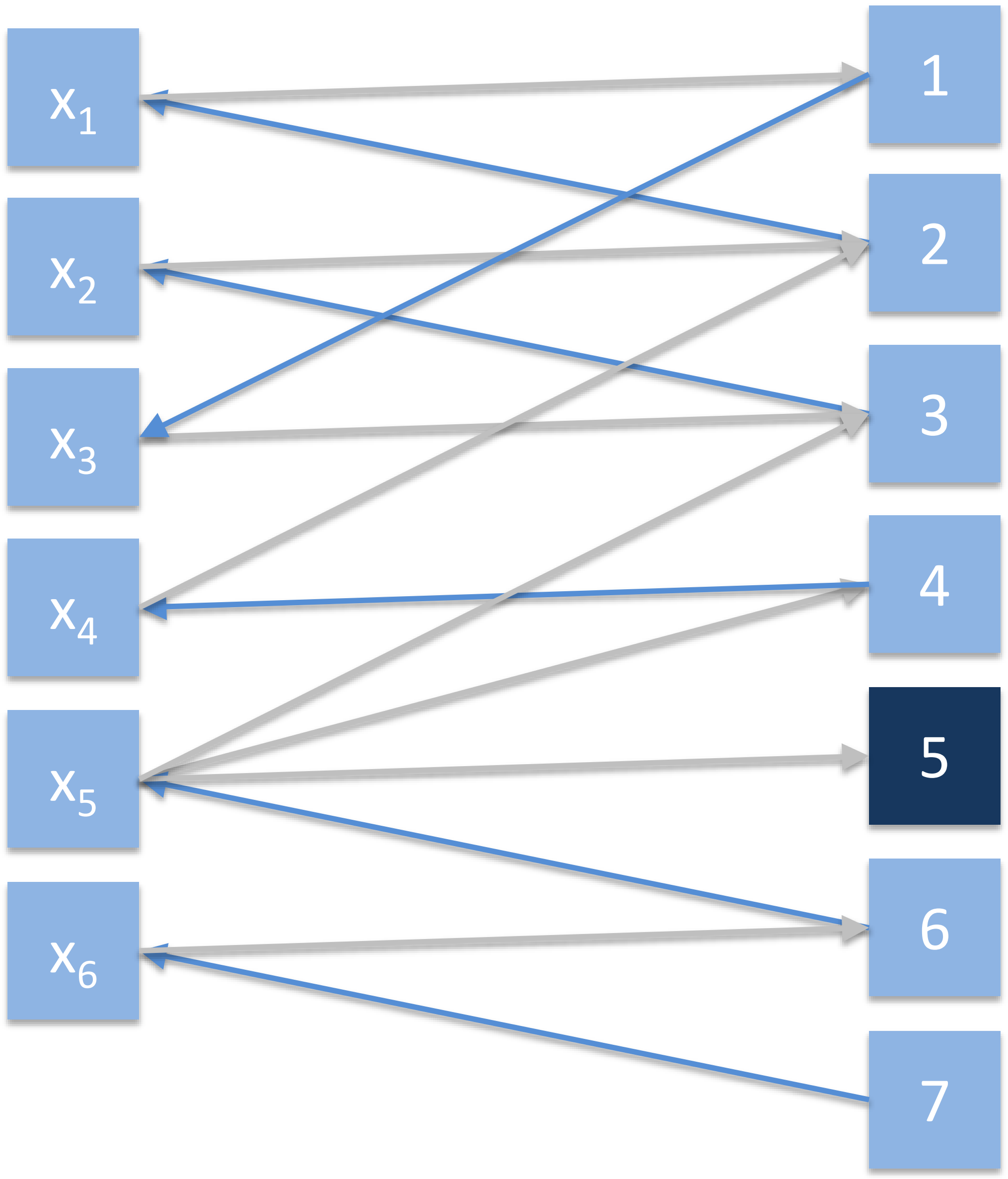
Directed graph

An alternating path

Change the direction

Repeat

Maximum matching



Directed graph

An alternating path

Change the direction

Repeat

Maximum matching

Feasibility of the Alldifferent constraint

- Use a bipartite graph
  - Vertex set for the variables
  - Vertex set for the values
  - Edge (x,v) if v is in D(x)
- Feasibility
  - Alldifferent is feasible iff the size of the maximum matching equals the number of variables
- Finding a maximum matching
  - Improve a matching using alternating paths in the directed graph obtained by the proper orientation of the edges

Alldifferent constraint: pruning

- The constraint
  - Alldifferent(x_1, …, x_n)
- How to prune?
  - v must be removed from the domain of x if the edge (x, v) appears in no maximum matching
- Only need to look at the edges not present in the maximum matching
- Naïve approach
  - Force the edge (x,v) in the matching, i.e., remove all other edges (x,w)
  - Search for a maximum matching
  - If it is smaller than the number of vars,

    v can be removed from D(x)

Alldifferent constraint: pruning

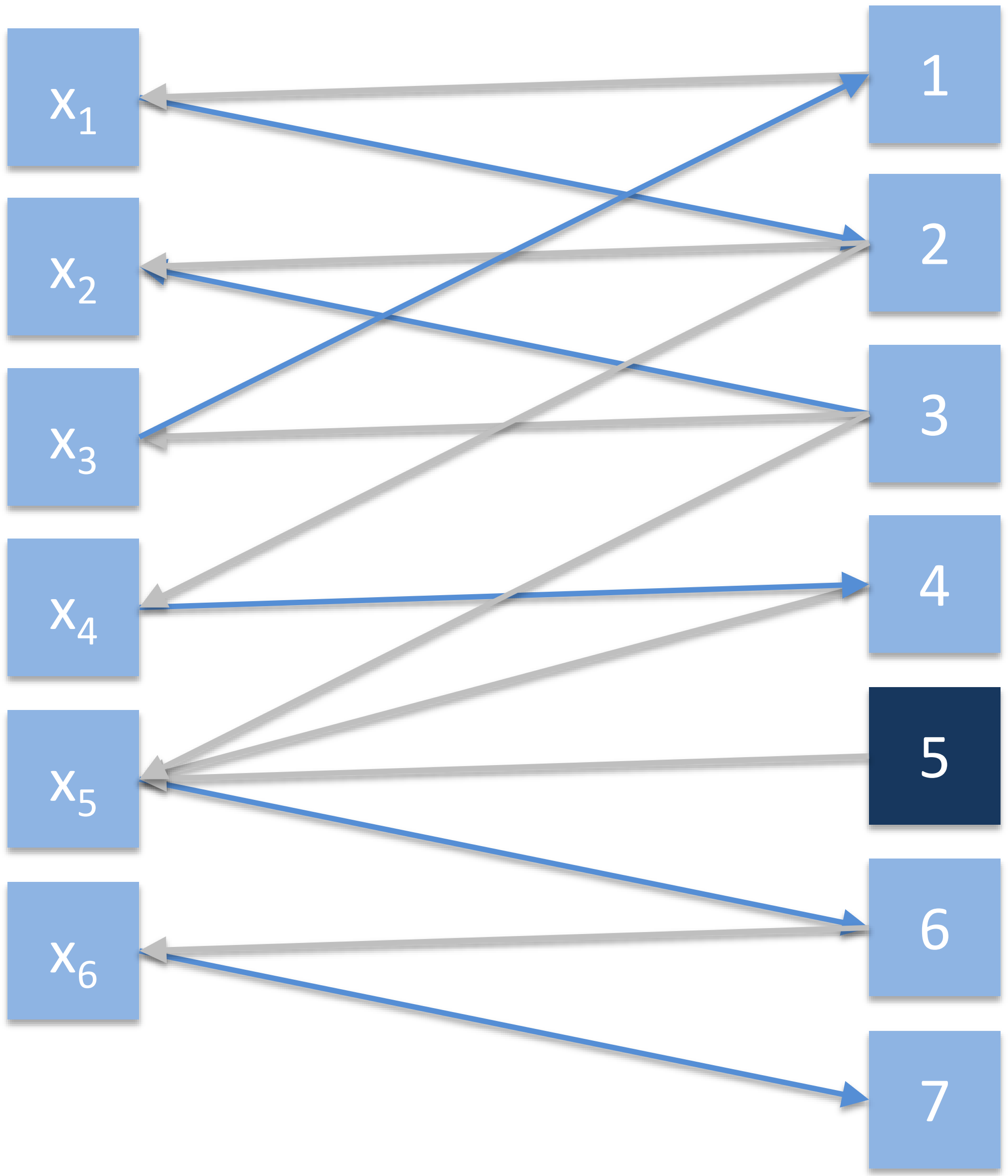
- Basic property (Berge, 1970)
  - An edge belongs to some but not all maximum matchings iff, given a maximum matching M, it belongs to either
    - An even alternative path starting at a free vertex
    - An even alternating cycle
- Note that
  - The edges not in the maximum matching do not belong to all maximum matchings
  - The above property tells us whether they belong to at least one maximum matching
  - The free vertices are the values

Alldifferent constraint: pruning

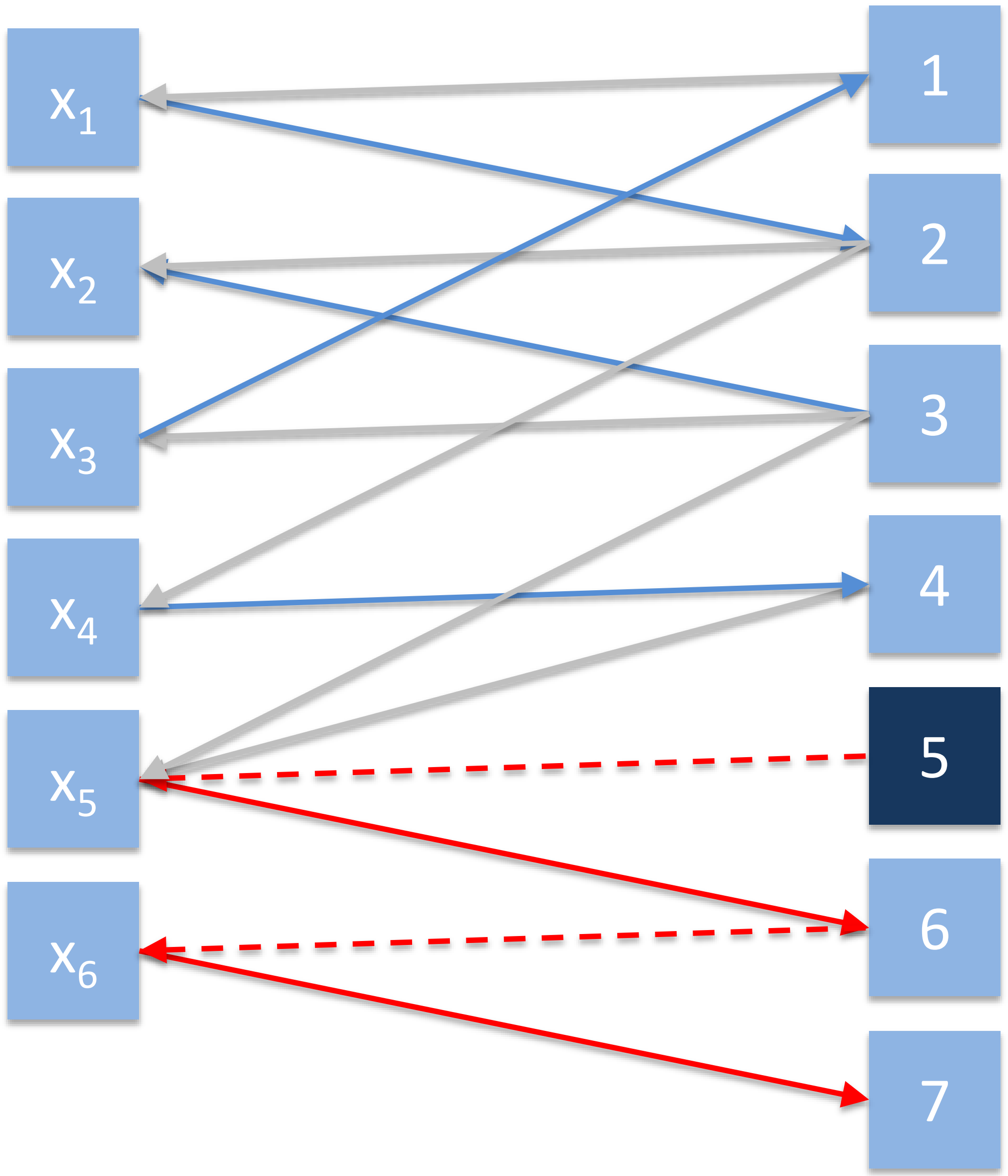
- Create a directed graph like before but reverse the direction of the edges

- Given a matching
  - Edges in the matching are oriented from left to right
  - Edges not in the matching are oriented from bottom to top

Maximum matching

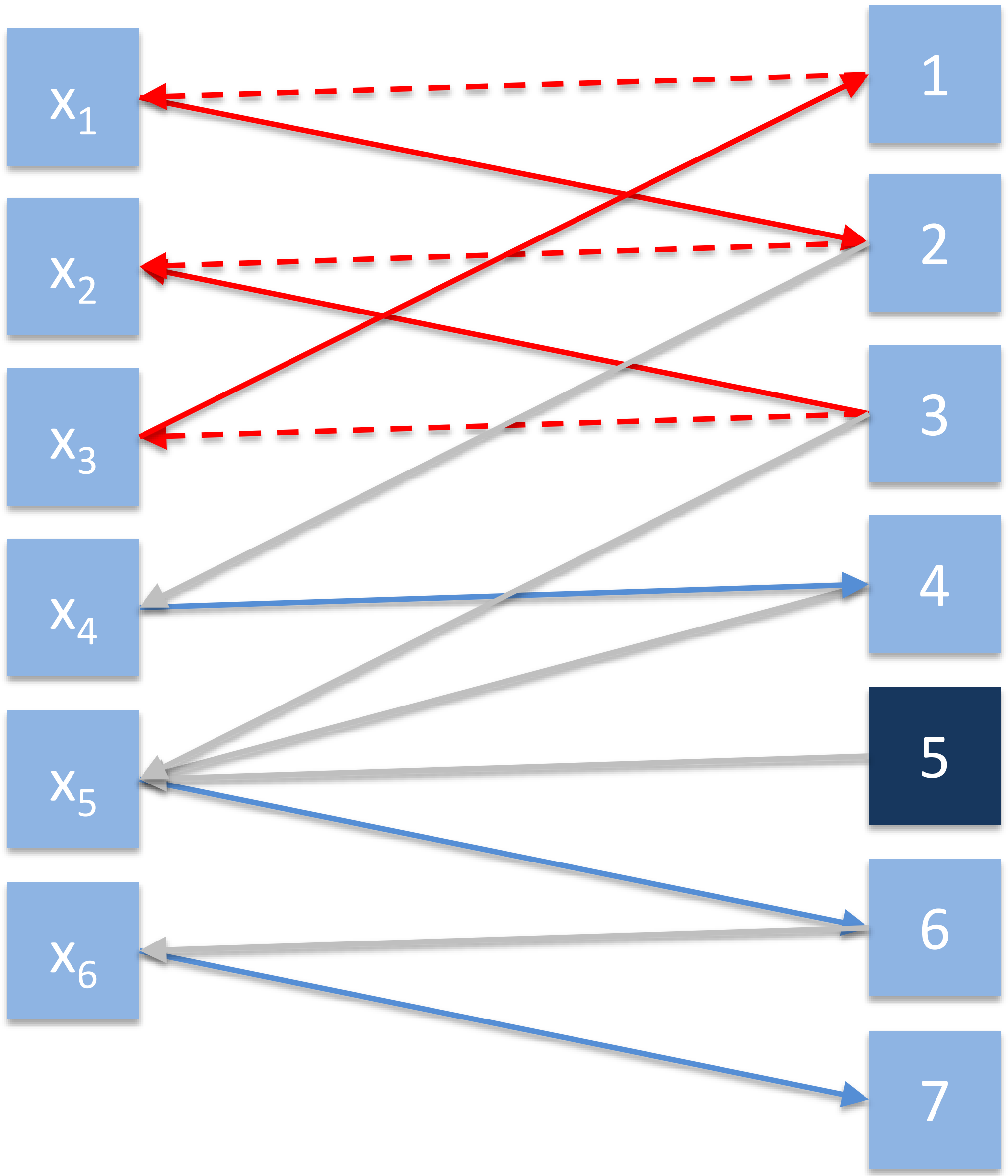


Maximum matching



An even alternative path starting at a free vertex

Maximum matching

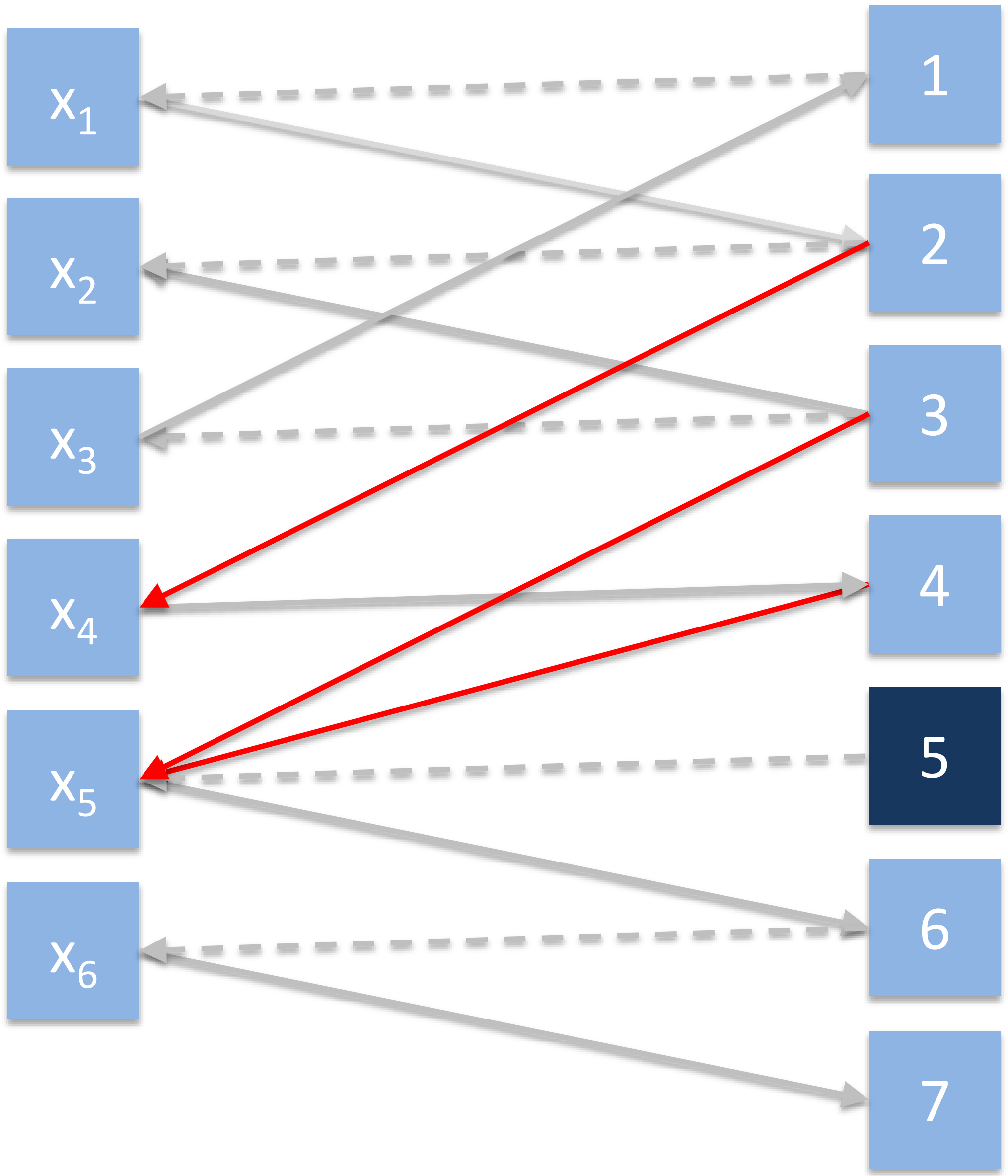


An even alternating cycle

Alldifferent constraint: pruning

- Given a maximum matching M, create a directed graph like before but reverse the direction of edges
- Search for even alternating path starting from a free label vertex: P
- Search for all loops and collect all the edges belonging to them: C
- Remove all edges belonging to M, P, C

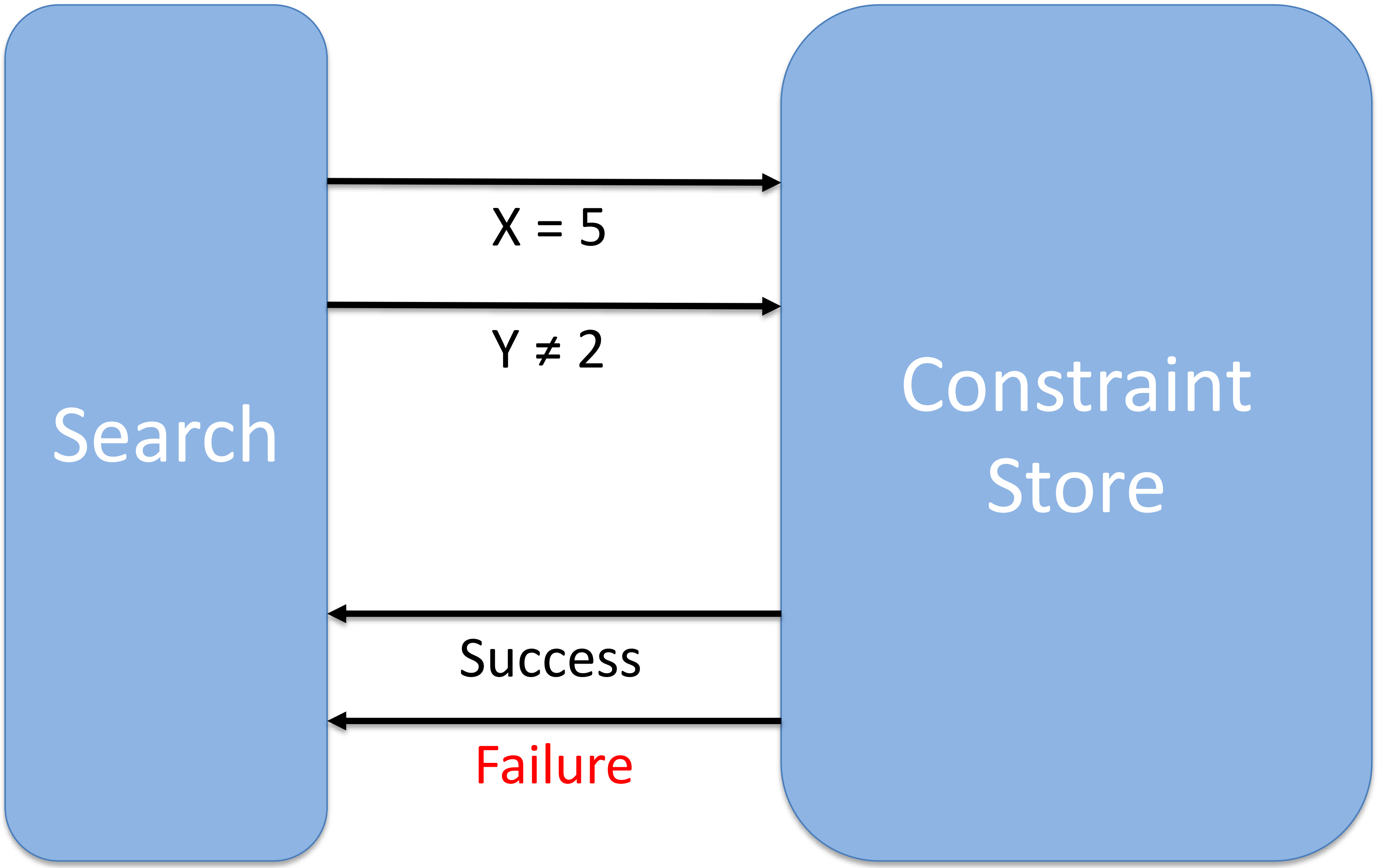
Maximum matching



Edges that can be removed

Search in constraint programming

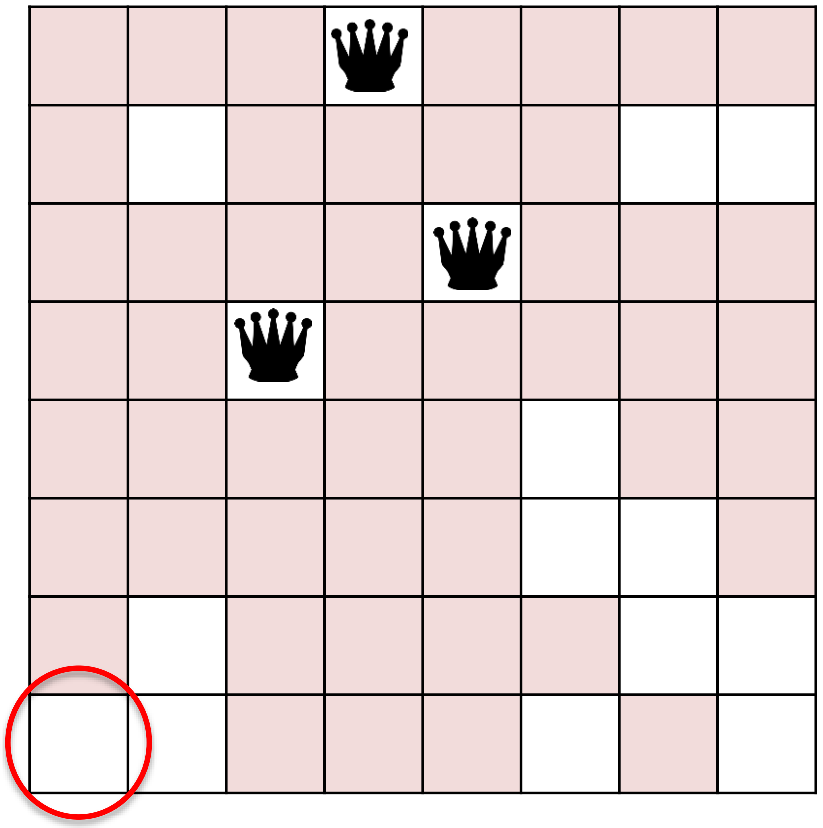
Computational paradigm



Search in constraint programming

- Key idea
  - Use feasibility information for branching
- First-fail principle
  - Try first where you are the most likely to fail
- Why first-fail principle?
  - Do not spend time doing easy stuff first and avoid redoing the difficult path
- The ultimate goal
  - Creating small search trees

First-fail search in 8 queens



Hard position
only one option

Trial and error

- When a constraint fails
    - that is, when adding a constraint to the constraint store returns a failure
- The solver goes back to the last guess
    - and assigns a value that has not been tried before
    - If no such value is left, the system backtracks to an earlier guessing instruction

Trial and error

```
range R = 1..8;
var{int} row[R] in R;
solve {
    forall(i in R,j in R: i < j) {
        row[i] ≠  row[j];
        row[i] ≠  row[j] + (j - i);
        row[i] ≠  row[j] - (j - i);
    }
}
using {
   forall(r in R)
       tryall(v in R)
          row[r] = v;
}
```

```
range R = 1..8;
var{int} row[R] in R;
solve {
    forall(i in R,j in R: i < j) {
        row[i] ≠  row[j];
        row[i] ≠  row[j] + (j - i);
        row[i] ≠  row[j] - (j - i);
    }
}
using {
    tryall(v in R) row[1] = v;
    tryall(v in R) row[2] = v;
    tryall(v in R) row[3] = v;
    tryall(v in R) row[4] = v;
    tryall(v in R) row[5] = v;
    tryall(v in R) row[6] = v;
    tryall(v in R) row[7] = v;
    tryall(v in R) row[8] = v;
}
```

Trial and error

```
range R = 1..8;
var{int} row[R] in R;
solve {
   forall(i in R,j in R: i < j) {
      row[i] ≠ row[j];
      row[i] ≠ row[j] + (j - i);
      row[i] ≠ row[j] - (j - i);
   }
}
using {
   forall(r in R)
      tryall(v in R)
         row[r] = v;
}
```

⇨

```
range R = 1..8;
var{int} row[R] in R;
solve {
   forall(i in R,j in R: i < j) {
      row[i] ≠ row[j];
      row[i] ≠ row[j] + (j - i);
      row[i] ≠ row[j] - (j - i);
   }
}
using {
   forall(r in R)
     try row[r] = 1;
     |   row[r] = 2;
     |   row[r] = 3;
     |   row[r] = 4;
     |   row[r] = 5;
     |   row[r] = 6;
     |   row[r] = 7;
     |   row[r] = 8;
     endtry;
}
```

Search strategies

Active research area. Some approaches:

- Variable/value labeling
- Value/variable labeling
- Symmetry breaking during search
- Randomization and restarts
- Domain splitting
- Focusing on the objective
- …
- …
- …

Variable/value labeling

- Two steps
  - Choose the variable to assign next
  - Choose the value to assign
- First-fail principle
  - Choose the variable with the smallest domain
- The variable ordering is dynamic
  - Reconsider the selection after each choice

Variable/value labeling

- Two steps
  - Choose the variable to assign next
  - Choose the value to assign
- First-fail principle
  - Choose the variable to with the smallest domain
  - Choose the most constrained variable
- Use a lexicographic criterion
  - First the domain size
  - Next the proximity to the middle of the board

Variable/value labeling

- Two steps
  - Choose the variable to assign next
  - Choose the value to assign

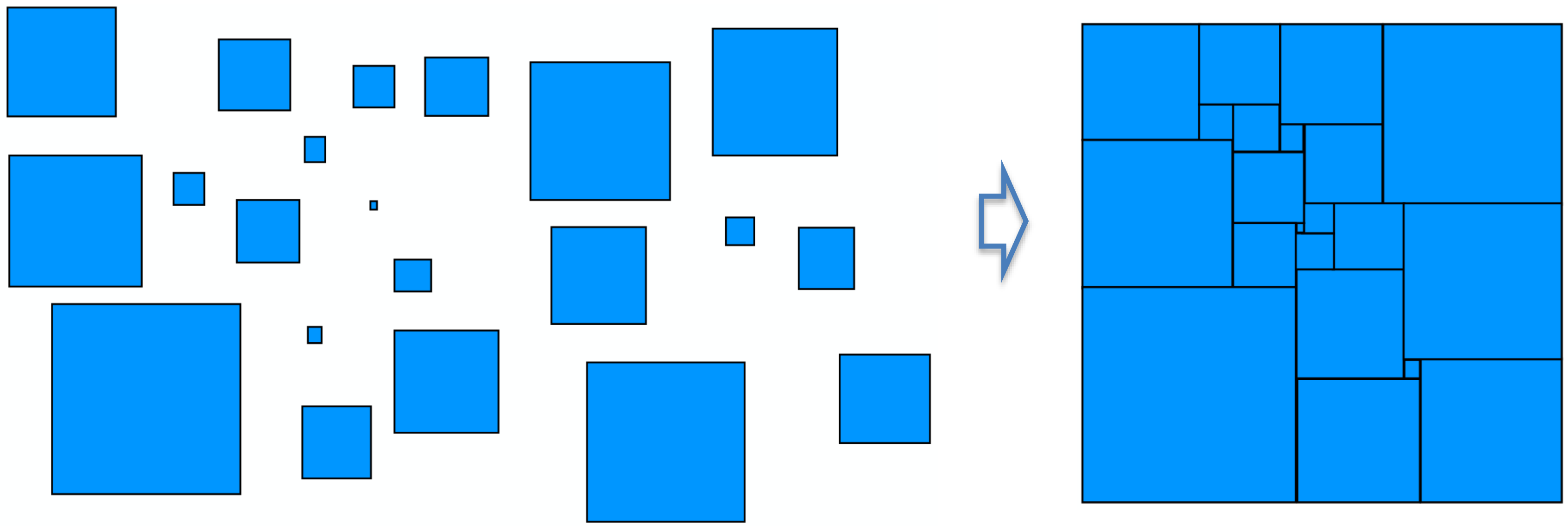
```
range R = 1..8;
var{int} row[R] in R;
solve {
   forall(i in R,j in R: i < j) {
      row[i] ≠  row[j];
      row[i] ≠  row[j] + (j - i);
      row[i] ≠  row[j] - (j - i);
   }
}
using {
   forall(r in R)
     by (row[r].getSize(),abs(r-n/2))
      tryall(v in R)
         row[r] = v;
}
```

Variable/value labeling

- Two steps
  - Choose the value to assign next
  - Choose the variable to assign to this value
- Why it is useful?
  - You may know that a value must be assigned
  - Often the case in scheduling and resource allocation problems

The perfect square problem

Task: arrange small squares into a big square



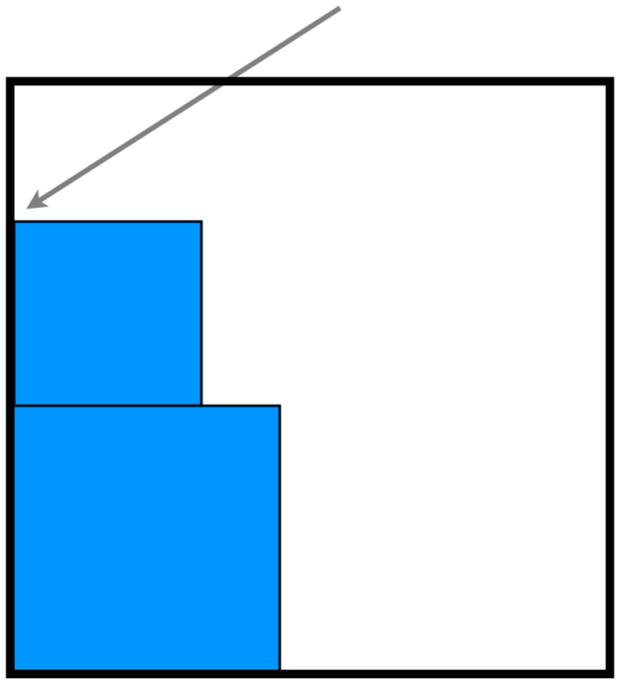
The perfect square problem

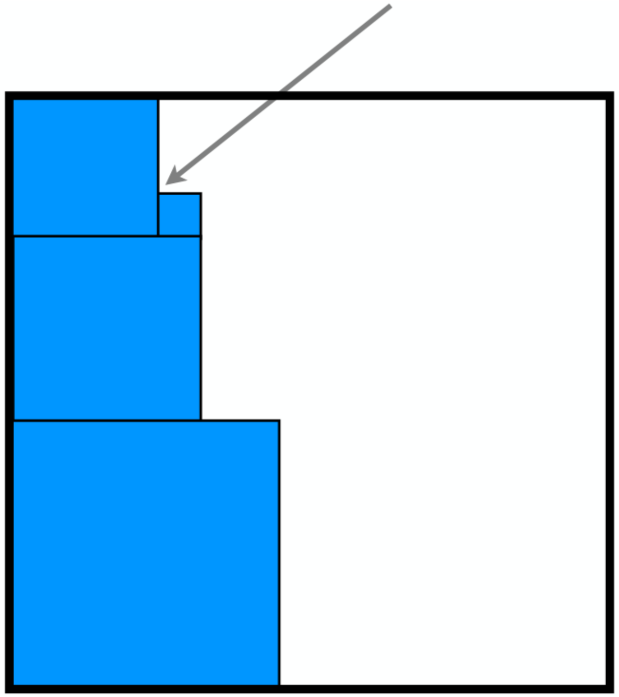
- What are the decision variables?
  - x and y-coordinates of the bottom-left corner of every square

- What are the constraints?
  - The squares fit in the larger square
  - The squares do not overlap

The value/variable labeling

- Why a value/variable labeling
  - We know that there is no empty space in the square to be filled





The value/variable labeling

- Why a value/variable labeling
  - We know that there is no empty space in the square to be filled
- What is the labeling doing?
  - Choose a x-coordinate p
  - For all square i, decide whether to place i at coordinate p
    - That is, whether the bottom-left corner of i has x-coordinate p
  - Repeat for all x-coordinates
  - Repeat for all y-coordinates

Symmetry breaking during search

Scene allocation problem

- Shooting scenes for a movie
  - an actor plays in some of the scenes
  - at most k scenes can be shot per day
  - each actor is paid by the day
  - scenes are different, actors are different
- Objective
  - Minimize the total cost
- Symmetries
  - What kind of symmetries do we have here?

Symmetry breaking: scene allocation

- Value symmetries
  - the days are interchangeable
- How do we eliminate these symmetries?
  - For each scene we consider only the used days and one new day
- Side effect
  - Interferes with the search heuristics
- Can we avoid this?
  - Symmetry-breaking during search
  - Dynamically impose the symmetry-breaking constraints
  - Same constraints, the order is different and discovered dynamically

Symmetry breaking: scene allocation

- Choose a scene to shoot
  - Use good heuristics
    - First-fail
    - Expensive scene
- Consider existing days + 1 new day
  - To label the scene
- Advantages
  - Break symmetries
  - Does not interfere with the search heuristics

Randomization and restarts

- Sometimes there is no obvious search ordering
  - But there exist some good ones

- How to find them?
  - Brute force
  - Randomization and restarts

- Key idea
  - Try a random ordering
  - If no solution is found after some limit, restart the search
  - and possibly increase the limit