ĐẠI HỌC

BÁCH KHOA

25 YEARS ANNIVERSARY

SOICT

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

FUNDAMENTALS OF OPTIMIZATION

Constraint Programming

CONTENT

- Constraint Satisfaction Optimization Problems
- Constraint Propagation
- Branching and Backtracking Search
- Examples

Constraint Satisfaction Problems

- Variables
  - $X = \{X_0, X_1, X_2, X_3, X_4\}$
- Domain
  - $X_0, X_1, X_2, X_3, X_4 \in \{1,2,3,4,5\}$
- Constraints
  - C_1: $X_2 + 3 \neq X_1$
  - C_2: $X_3 \leq X_4$
  - C_3: $X_2 + X_3 = X_0 + 1$
  - C_4: $X_4 \leq 3$
  - C_5: $X_1 + X_4 = 7$
  - C_6: $X_2 = 1 \Rightarrow X_4 \neq 2$

Constraint Satisfaction Problems

- CSP = (X,D,C), in which:
  - $X = \{X_1,...,X_N\}$ – set of variables
  - $D = \{D(X_1),\ldots, D(X_N)\}$ – domains of variables
  - $C = \{C_1,..,C_K\}$ – set of constraints over variables
  - Denote $X(c)$ – set of variables appearing in the constraint c

Constraint Satisfaction Problems

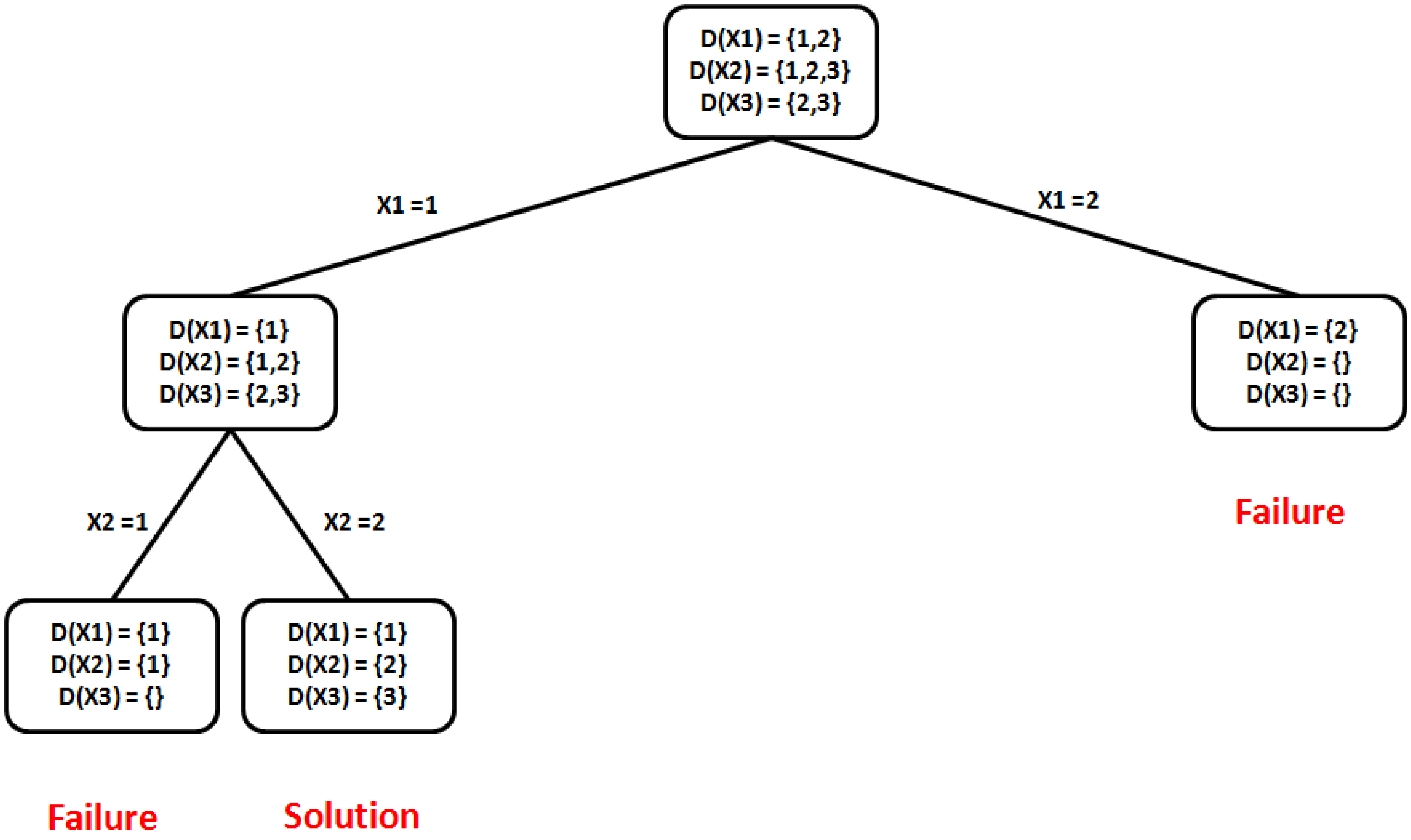
- COP = (X,D,C,f), in which:
  - $X = \{X_1,...,X_N\}$ – set of variables
  - $D = \{D(X_1),\ldots, D(X_N)\}$ – domains of variables
  - $C = \{C_1,..,C_K\}$ – set of constraints over variables
  - Denote $X(c)$ – set of variables appearing in the constraint c
  - f: objective function to be optimized

Constraint Programming

- A computation paradigm for solving CSP, COP combining
  - Constraint Propagation: narrow the search space by pruning redundant values from the domains of variables
  - Branching (backtracking search): split the problem into equivalent sub-problems by
    - Instantiating some variables with values of its domain
    - Split the domain of a selected variable into sub-domains

Constraint Programming

X = {X1,X2,X3}
D(X1) = {1,2,3,4}, D(X2) = {1,2,3}, D(X3) = {1,2,3}
C1: X1 ≤ X2
C2: X3 = X1+X2
C3: X2+X3 = 5



Constraint Propagation

- Domain consistency (DC)
  - Given a CSP = (X,D,C), a constraint $c \in C$ is called domain consistent if for each variable $X_i \in X(c)$ and each value $v \in D(X_i)$, there exists values for variables of $X(c) \setminus \{X_i\}$ such that c is satisfied
  - A CSP is called domain consistent if c is domain consistent for all $c \in C$

Constraint Propagation

- DC algorithms aim at pruning redundant values from the domains of variables so that the obtained equivalent CSP is domain consistent

Constraint Propagation

- Example: CSP = (X,D,C) in which:
  - $X = \{X_1, X_2, X_3, X_4\}$
  - $D(X_1) = \{1,2,3,4\}$, $D(X_2) =\{1,2,3,4,5,6,7\}$, $D(X_3) = \{2,3,4,5\}$, $D(X_4) = \{1,2,3,4,5,6\}$
  - $C = \{c_1,c_2,c_3\}$ với
    - $c_1 \equiv X_1 + X_2 \geq 5$
    - $c_2 \equiv X_1 + X_3 \geq X_4$
    - $c_3 \equiv X_1 + 3 \geq X_3$

→ CSP is domain consistent

- When branching, consider $X_1 = 1$, a DC algorithm will transform the given CSP to an equivalent domain consistent CSP[1] having : $D^1(X_1) = \{1\}$, $D^1(X_2) = \{4,5,6,7\}$, $D^1(X_3) = \{2,3,4\}$, $D^1(X_4) = \{1,2,3,4,5\}$

Constraint Propagation

- A domain consistent CSP does not ensure to have feasible solutions
- Example:
  - $X = \{X_1, X_2, X_3\}$
  - $D(X_1) = D(X_2) = D(X_3) = \{0,1\}$
  - $c_1 \equiv X_1 \neq X_2$, $c_2 \equiv X_1 \neq X_3$, $c_3 \equiv X_2 \neq X_3$

  → The CSP is domain consistent but does not have any feasible solution

Constraint Propagation

```
Algorithm AC3(X,D,C){
  $Q = \{(x,c) \mid c \in C \wedge x \in X(c)\}$;
  while(Q not empty){
    select and remove (x,c) from Q;
    if ReviseAC3(x,c) then{
      if D(x) = {} then
        return false;
      else
        $Q = Q \cup \{(x',c') \mid c' \in C \setminus \{c\} \wedge x,x' \in X(c') \wedge x \neq x'\}$
    }
  }
  return true;
}
```

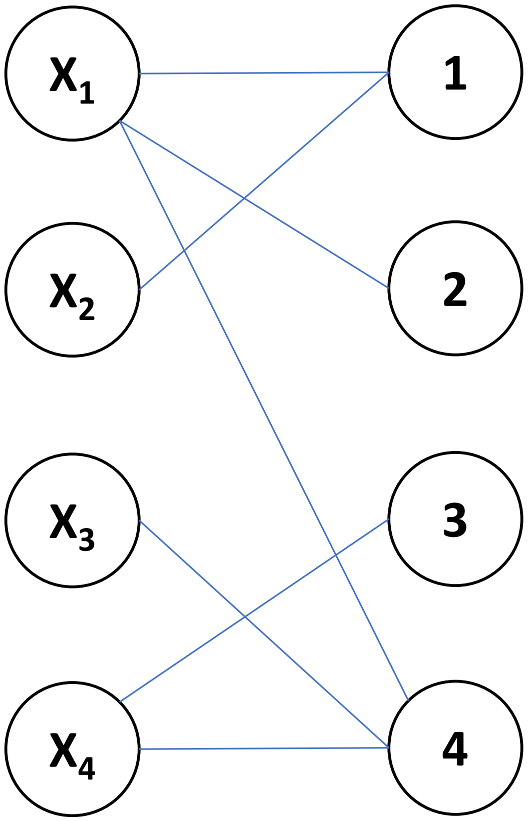
```
Algorithm ReviseAC3(x,c){
  CHANGE = false;
  for $v \in D(x)$ do{
    if there does not exists other values
      of $X(c) \setminus \{x\}$ such that c
        is satisfied then{
      remove v from D(x);
      CHANGE = true;
    }
  }
  return CHANGE;
}
```

Constraint Propagation

- Some constraints, e.g., binary constraints (related 2 variables) → have efficient DC algorithm
- Constraint AllDifferent(X_1,X_2,…,X_N), the DC algorithm is efficient based on the matching (Max-Matching) algorithm on bipartite graphs
  - Nodes on the right-hand side are variables and nodes on the left-hand side are values
  - For each edge (X_i, v), (với $v \in D(X_i)$), if there does not exist a matching of size N containing (X_i, v), then v is removed from $D(X_i)$

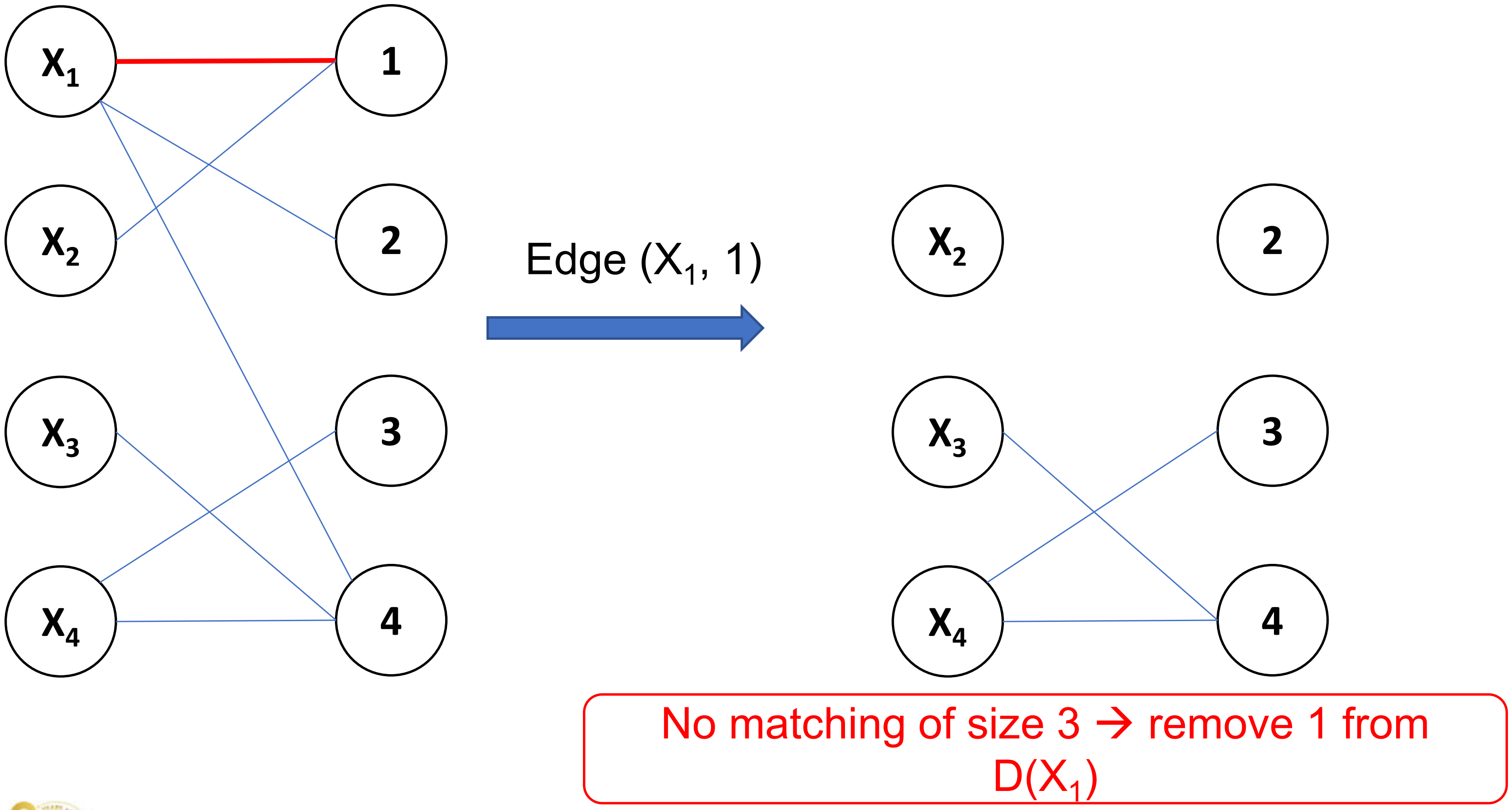
Constraint Propagation

- $X = \{X_1, X_2, X_3, X_4\}$
- $D(X_1) = \{1,2,4\}$, $D(X_2) = \{1\}$, $D(X_3) = \{4\}$, $D(X_4) = \{3,4\}$



Constraint Propagation

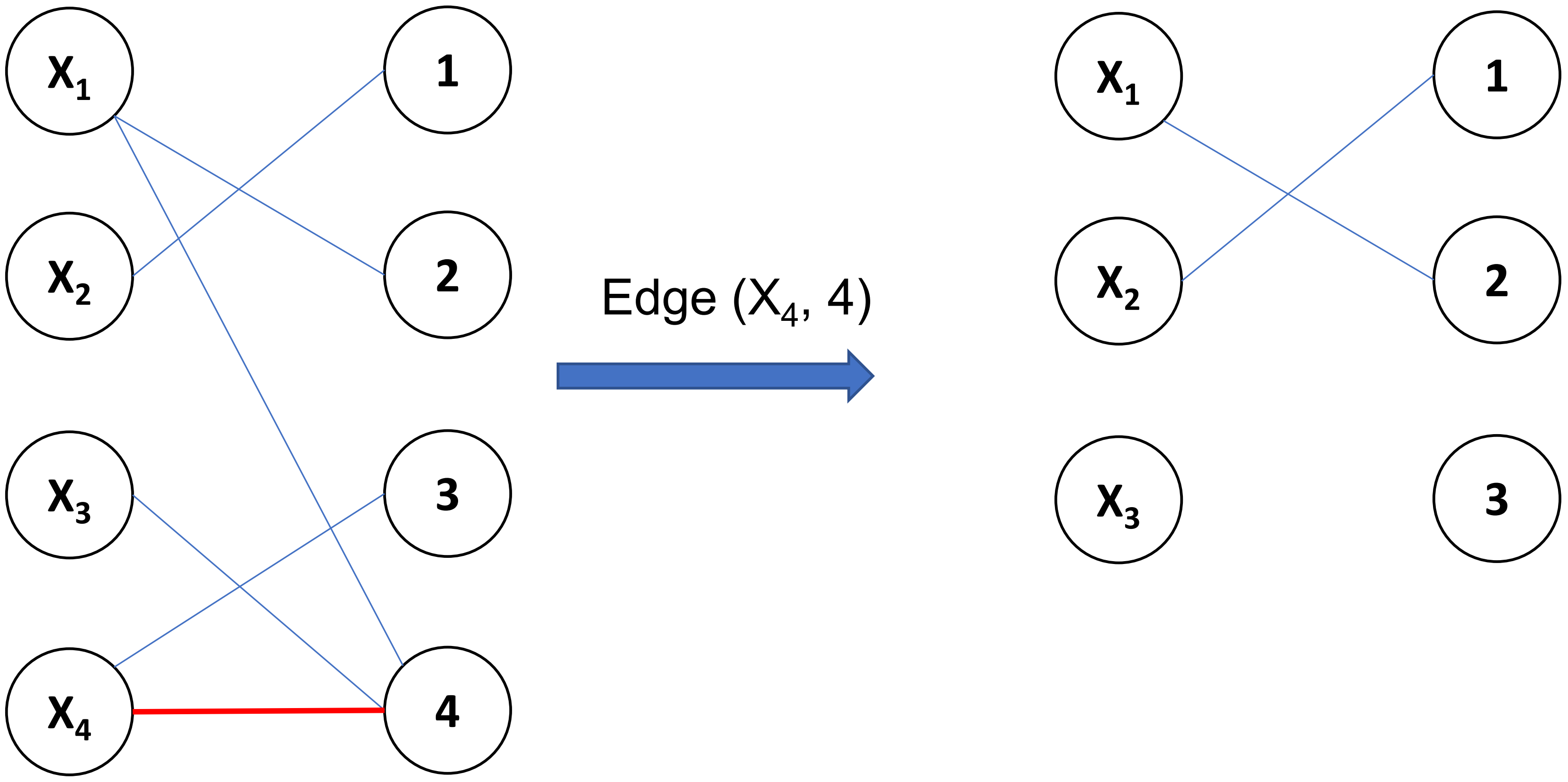
- $X = \{X_1, X_2, X_3, X_4\}$
- $D(X_1) = \{1,2,4\}$, $D(X_2) = \{1\}$, $D(X_3) = \{4\}$, $D(X_4) = \{3,4\}$



No matching of size 3 → remove 1 from $D(X_1)$

Constraint Propagation

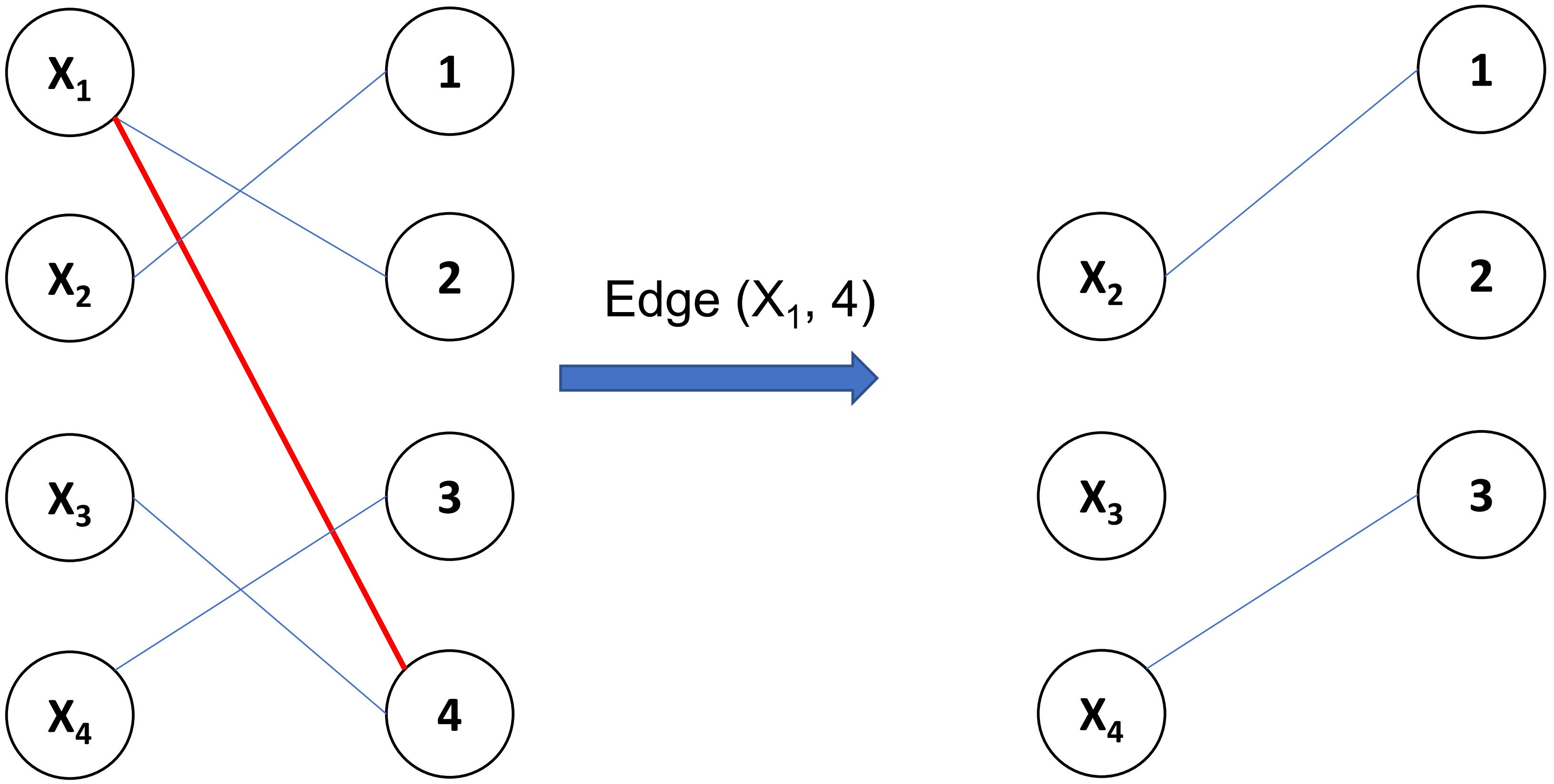
- $X = \{X_1, X_2, X_3, X_4\}$
- $D(X_1) = \{2,4\}$, $D(X_2) = \{1\}$, $D(X_3) = \{4\}$, $D(X_4) = \{3,4\}$



No matching of size 3 → removed 4 from $D(X_4)$

Constraint Propagation

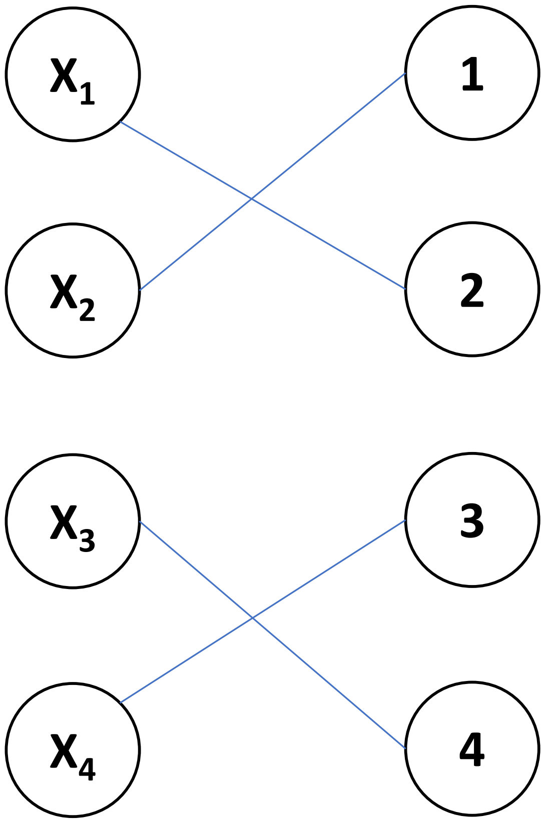
- $X = \{X_1, X_2, X_3, X_4\}$
- $D(X_1) = \{2,4\}$, $D(X_2) = \{1\}$, $D(X_3) = \{4\}$, $D(X_4) = \{3\}$



No matching of size 3 → removed 4 from $D(X_1)$

Constraint Propagation

- $X = \{X_1, X_2, X_3, X_4\}$
- $D(X_1) = \{2\}$, $D(X_2) = \{1\}$, $D(X_3) = \{4\}$, $D(X_4) = \{3\}$

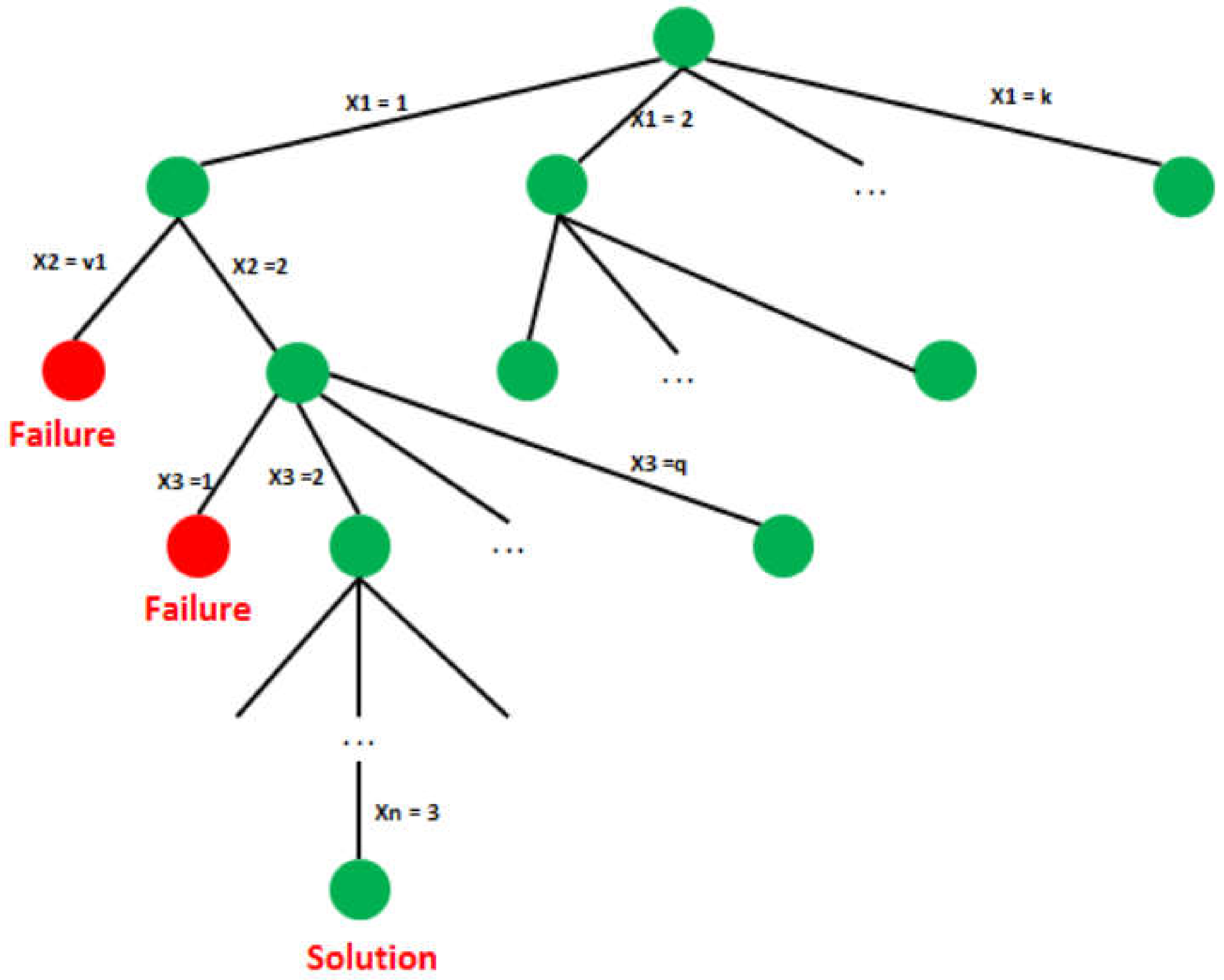


Feasible solution!

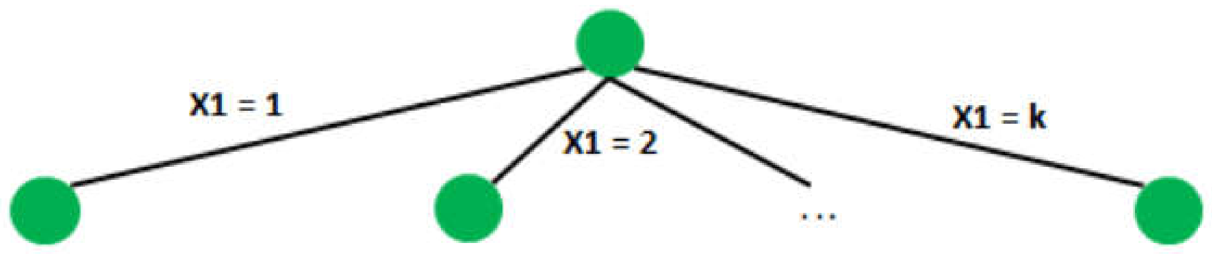
Branching & Backtracking Search

- Constraint propagation is not enough for finding feasible solutions
- Combine constraint propagation with branching and backtracking search
  - Split the original CSP P_0 into sub-problems CSP $P_1,\dots,P_M$
    - Set of solutions of P_0 is equivalent to the union of sets of solutions to $P_1,\dots,P_M$
    - Domain of each variable in $P_1,\dots,P_M$ is not greater than the domain of that variable in P_0
  - Search Tree
    - Root is the original CSP P_0
    - Each node of the tree is a CSP
    - If $P_1,\dots,P_M$ are children of P_0 then the set of solutions of P_0 is equivalent to the union of sets of solutions to $P_1,\dots,P_M$
    - Leaves
      - A feasible solution
      - Failure (a variable has an empty domain)

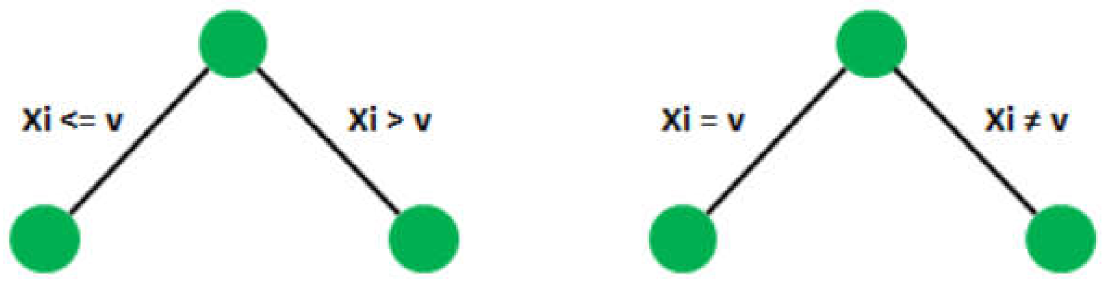
Branching & Backtracking Search



Branching & Backtracking Search



Enumeration



Splitting a domain

Branching & Backtracking Search

- Search strategies
  - Variable selection
    - **dom** heuristic: select a variable having the smallest domain
    - **deg** heuristic: select a variable participating in most of the constraints
    - **dom+deg** heuristic: first apply **dom**, then use **deg** when tie break (when there are more than one variable with the same smallest domain size)
    - **dom/deg**: select a variable having the smallest dom/deg
  - Value selection
    - Increasing order
    - Decreasing order

Example

- Variables
  - $X = \{X_0, X_1, X_2, X_3, X_4\}$
- Domain
  - $X_0, X_1, X_2, X_3, X_4 \in \{1,2,3,4,5\}$
- Constraints
  - C_1: $X_2 + 3 \neq X_1$
  - C_2: $X_3 \leq X_4$
  - C_3: $X_2 + X_3 = X_0 + 1$
  - C_4: $X_4 \leq 3$
  - C_5: $X_1 + X_4 = 7$
  - C_6: $X_2 = 1 \Rightarrow X_4 \neq 2$

Example

```
'''
If-Then-Else expression
if x[2] = 1 then x[4] != 2
'''
from ortools.sat.python import cp_model
class VarArraySolutionPrinter(cp_model.CpSolverSolutionCallback):
          #print intermediate solution
          def __init__(self,variables):
                    cp_model.CpSolverSolutionCallback.__init__(self)
                    self.__variables = variables
                    self.__solution_count = 0
          def on_solution_callback(self):
                    self.__solution_count += 1
                    for v in self.__variables:
                              print('%s = %i'% (v,self.Value(v)), end = ' ')
                    print()
          def solution_count():
                    return self.__solution_count
```

Example

```
model = cp_model.CpModel()
x = {}
for i in range(5):
           x[i] = model.NewIntVar(1,5,'x[' + str(i) + ']')

c1 = model.Add(x[2] + 3 != x[1])
c2 = model.Add(x[3] <= x[4])
c3 = model.Add(x[2] + x[3] == x[0] + 1)
c4 = model.Add(x[4] <= 3)
c5 = model.Add(x[1] + x[4] == 7)
b = model.NewBoolVar('b')
#constraints
model.Add(x[2] == 1).OnlyEnforceIf(b)
model.Add(x[2] != 1).OnlyEnforceIf(b.Not())
model.Add(x[4] != 2).OnlyEnforceIf(b)
```

Example

```
solver = cp_model.CpSolver()

#Force the solver to follow the decision strategy exactly
solver.parameters.search_branching = cp_model.FIXED_SEARCH

vars = [x[i] for i in range(5)]


solution_printer = VarArraySolutionPrinter(vars)
solver.SearchForAllSolutions(model,solution_printer)
```

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Thank you for your attentions!