



ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

TỐI ƯU LẬP KẾ HOẠCH

Quy hoạch ràng buộc

Nội dung

- Tổng quan phương pháp quy hoạch ràng buộc
- Thành phần trong quy hoạch ràng buộc
- Thu hẹp không gian tìm kiếm
- Phân nhánh và tìm kiếm quay lui
- Ví dụ minh họa bài toán N-queens
- Bài toán Sudoku
- Bài toán phân bổ môn học

Tổng quan Quy hoạch ràng buộc

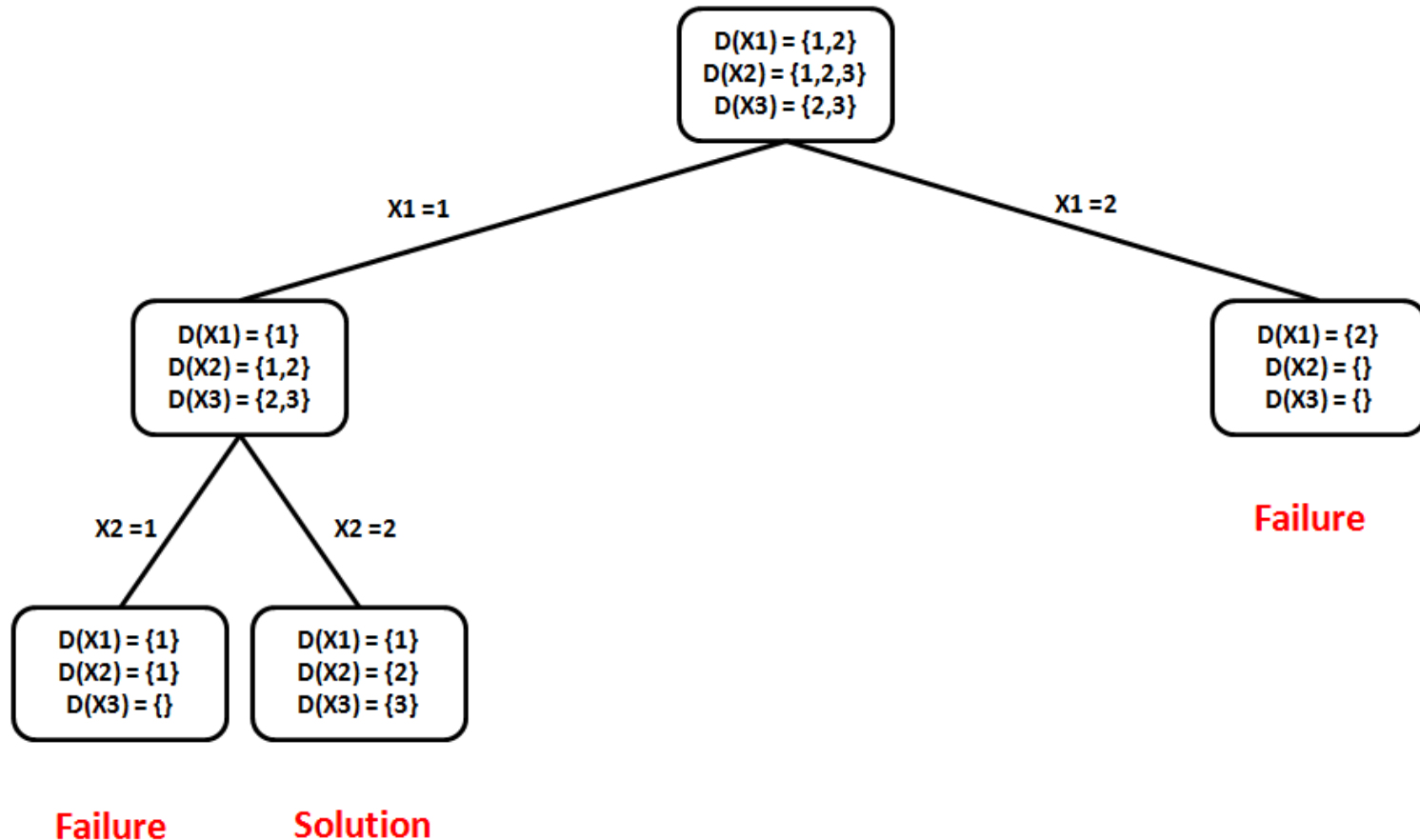
- Quy hoạch ràng buộc: Constraint Programming (hay CP)
- Bài toán tối ưu $COP = (X, D, C, f)$
 - $X = \{X_1, \dots, X_N\}$: tập các biến
 - $D = \{D_1, \dots, D_N\}$: miền giá trị của các biến
 - $C = \{C_1, \dots, C_K\}$: tập các ràng buộc
 - f : hàm mục tiêu
- CP: Một phương pháp giải đúng bài toán tối ưu tổ hợp
 - Dùng các ràng buộc để tĩa không gian tìm kiếm: loại bỏ các giá trị thừa ra khỏi miền giá trị của các biến
 - Phân nhánh và tìm kiếm quay lui: Chia không gian tìm kiếm thành các không gian con
 - Liệt kê các giá trị cho biến được lựa chọn
 - Phân hoạch tập giá trị của mỗi biến được lựa chọn thành 2 hoặc nhiều tập con

Tổng quan Quy hoạch ràng buộc

- Biến
 - $X = \{X_0, X_1, X_2, X_3, X_4\}$
- Miền giá trị
 - $X_0, X_1, X_2, X_3, X_4 \in \{1, 2, 3, 4, 5\}$
- Ràng buộc
 - $C_1: X_2 + 3 \neq X_1$
 - $C_2: X_3 \leq X_4$
 - $C_3: X_2 + X_3 = X_0 + 1$
 - $C_4: X_4 \leq 3$
 - $C_5: X_1 + X_4 = 7$
 - $C_6: X_2 = 1 \Rightarrow X_4 \neq 2$

Tổng quan Quy hoạch ràng buộc

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



Constraint Programming in a Nutshell

SEND

MORE

MONEY

Constraint Programming in a Nutshell

SEND + MORE = MONEY

$$\text{SEND} + \text{MORE} = \text{MONEY}$$

Assign distinct digits to the letters

S, E, N, D, M, O, R, Y

such that

$$\begin{array}{r} \text{S E N D} \\ + \text{M O R E} \\ \hline = \text{M O N E Y} \end{array}$$

holds.

$$\text{SEND} + \text{MORE} = \text{MONEY}$$

Assign distinct digits to the letters

S, E, N, D, M, O, R, Y

such that

$$\begin{array}{r} \text{S E N D} \\ + \text{M O R E} \\ \hline \end{array}$$

$$= \text{M O N E Y}$$

holds.

Solution

$$\begin{array}{r} 9567 \\ + 1085 \\ \hline \end{array}$$

$$= 10652$$

A Model for **MONEY**

- number of variables: 8
- constraints:

$$c_1 = \{ (S, E, N, D, M, O, R, Y) \in \mathbb{Z}^8 \mid 0 \leq S, \dots, Y \leq 9 \}$$

$$\begin{aligned} c_2 = \{ (S, E, N, D, M, O, R, Y) \in \mathbb{Z}^8 \mid \\ & 1000*S + 100*E + 10*N + D \\ & + 1000*M + 100*O + 10*R + E \\ & = 10000*M + 1000*O + 100*N + 10*E + Y \} \end{aligned}$$

A Model for **MONEY** (continued)

- more constraints

$$c_3 = \{ (S, E, N, D, M, O, R, Y) \in Z^8 \mid S \neq 0 \}$$

$$c_4 = \{ (S, E, N, D, M, O, R, Y) \in Z^8 \mid M \neq 0 \}$$

$$c_5 = \{ (S, E, N, D, M, O, R, Y) \in Z^8 \mid S \dots Y \text{ all different} \}$$

Solution for **MONEY**

$$c_1 = \{ (S, E, N, D, M, O, R, Y) \in Z^8 \mid 0 \leq S, \dots, Y \leq 9 \}$$

$$\begin{aligned} c_2 = \{ (S, E, N, D, M, O, R, Y) \in Z^8 \mid \\ & 1000*S + 100*E + 10*N + D \\ & + 1000*M + 100*O + 10*R + E \\ & = 10000*M + 1000*O + 100*N + 10*E + Y \} \end{aligned}$$

$$c_3 = \{ (S, E, N, D, M, O, R, Y) \in Z^8 \mid S \neq 0 \}$$

$$c_4 = \{ (S, E, N, D, M, O, R, Y) \in Z^8 \mid M \neq 0 \}$$

$$c_5 = \{ (S, E, N, D, M, O, R, Y) \in Z^8 \mid S \dots Y \text{ all different} \}$$

Solution: $(9, 5, 6, 7, 1, 0, 8, 2) \in Z^8$

Elements of Constraint Programming

Exploiting constraints during tree search

- Use propagation algorithms for constraints
- Employ branching algorithm
- Execute exploration algorithm

S ∈ □

E ∈ □

N ∈ □

D ∈ □

M ∈ □

O ∈ □

R ∈ □

Y ∈ □

S E N D
+ M O R E

= M O N E Y

$0 \leq S, \dots, Y \leq 9$

$S \neq 0$

$M \neq 0$

S...Y all different

$$\begin{aligned} & 1000*S + 100*E + 10*N + D \\ & + 1000*M + 100*O + 10*R + E \\ & = 10000*M + 1000*O + 100*N + 10*E + Y \end{aligned}$$

Propagate

S \in {0..9}

E \in {0..9}

N \in {0..9}

D \in {0..9}

M \in {0..9}

O \in {0..9}

R \in {0..9}

Y \in {0..9}

S E N D
+ M O R E

= M O N E Y

$0 \leq S, \dots, Y \leq 9$

$S \neq 0$

$M \neq 0$

S...Y all different

$$\begin{aligned} & 1000*S + 100*E + 10*N + D \\ & + 1000*M + 100*O + 10*R + E \\ & = 10000*M + 1000*O + 100*N + 10*E + Y \end{aligned}$$

Propagate

$S \in \{1..9\}$

$E \in \{0..9\}$

$N \in \{0..9\}$

$D \in \{0..9\}$

$M \in \{1..9\}$

$O \in \{0..9\}$

$R \in \{0..9\}$

$Y \in \{0..9\}$

S E N D
+ M O R E

= M O N E Y

$0 \leq S, \dots, Y \leq 9$

$S \neq 0$

$M \neq 0$

S...Y all different

$$\begin{aligned} & 1000*S + 100*E + 10*N + D \\ & + 1000*M + 100*O + 10*R + E \\ & = 10000*M + 1000*O + 100*N + 10*E + Y \end{aligned}$$

Propagate

$S \in \{9\}$

$E \in \{4..7\}$

$N \in \{5..8\}$

$D \in \{2..8\}$

$M \in \{1\}$

$O \in \{0\}$

$R \in \{2..8\}$

$Y \in \{2..8\}$

S E N D
+ M O R E

= M O N E Y

$0 \leq S, \dots, Y \leq 9$

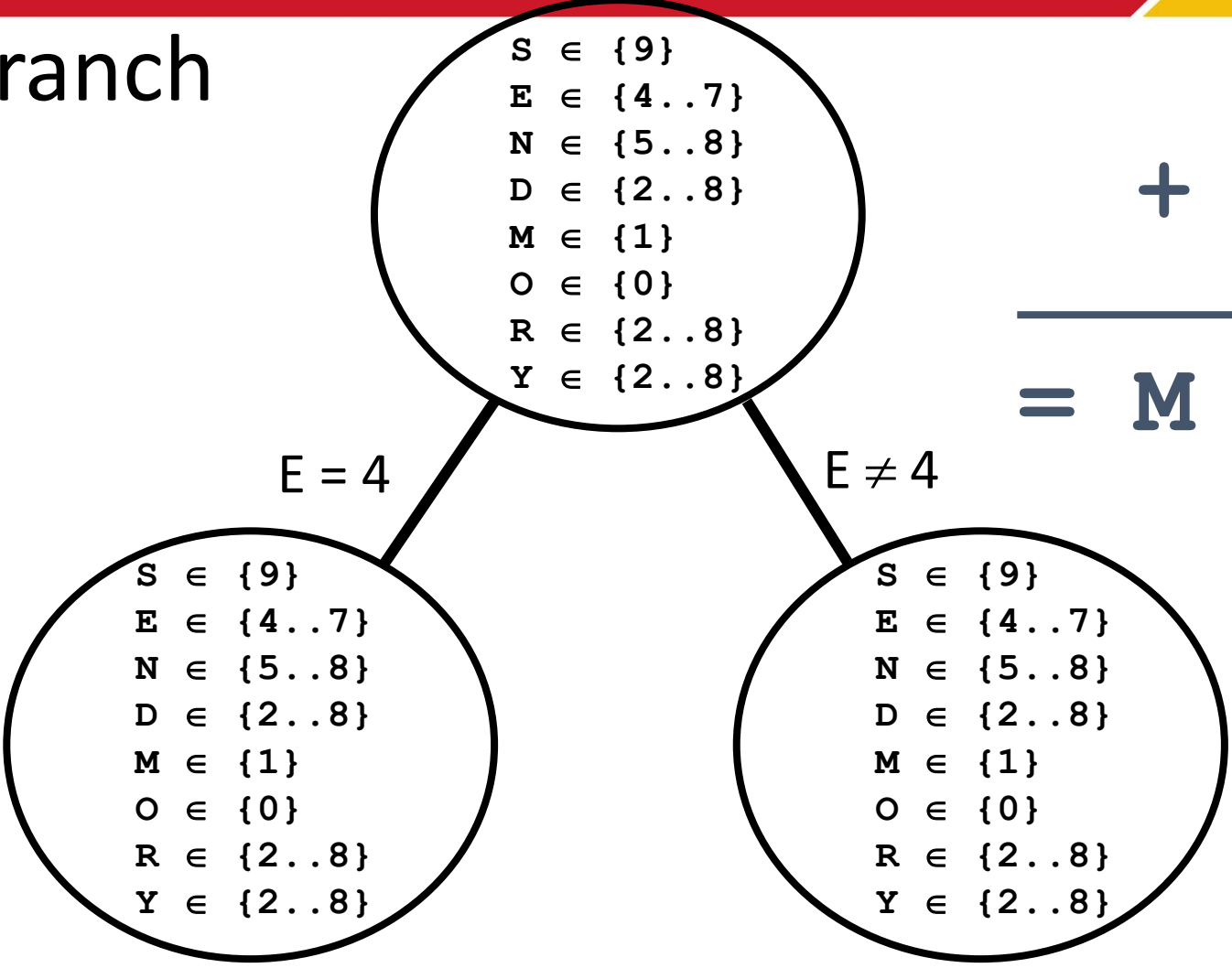
$S \neq 0$

$M \neq 0$

S...Y all different

$$\begin{aligned} & 1000*S + 100*E + 10*N + D \\ & + 1000*M + 100*O + 10*R + E \\ & = 10000*M + 1000*O + 100*N + 10*E + Y \end{aligned}$$

Branch



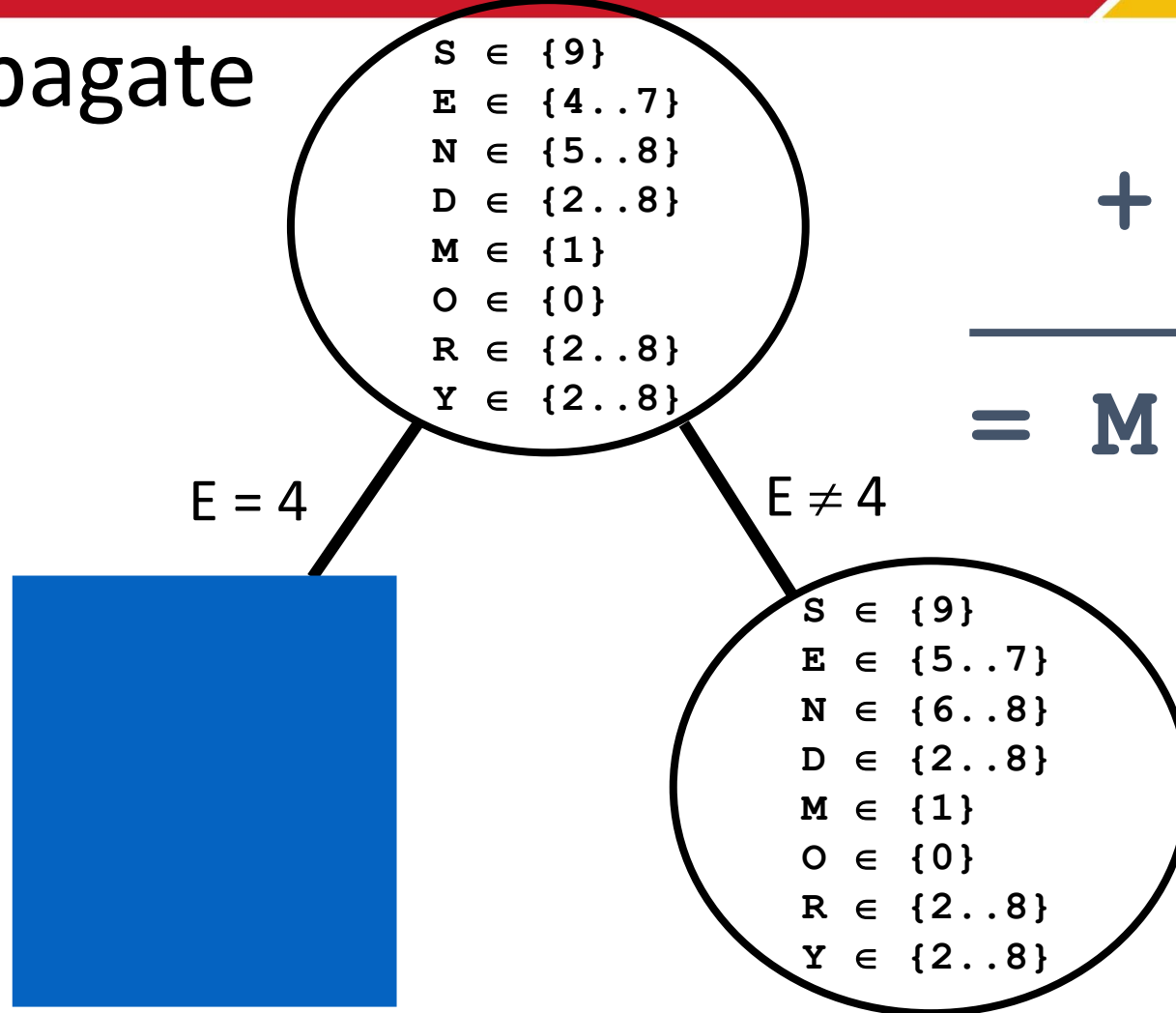
$$\begin{array}{r} \text{S E N D} \\ + \text{M O R E} \\ \hline = \text{M O N E Y} \end{array}$$

$0 \leq S, \dots, Y \leq 9$
 $S \neq 0$
 $M \neq 0$

S...Y all different

$$\begin{aligned} &1000*S + 100*E + 10*N + D \\ &+ 1000*M + 100*O + 10*R + E \\ &= 10000*M + 1000*O + 100*N + 10*E + Y \end{aligned}$$

Propagate



S E N D
+ M O R E

= M O N E Y

$0 \leq S, \dots, Y \leq 9$

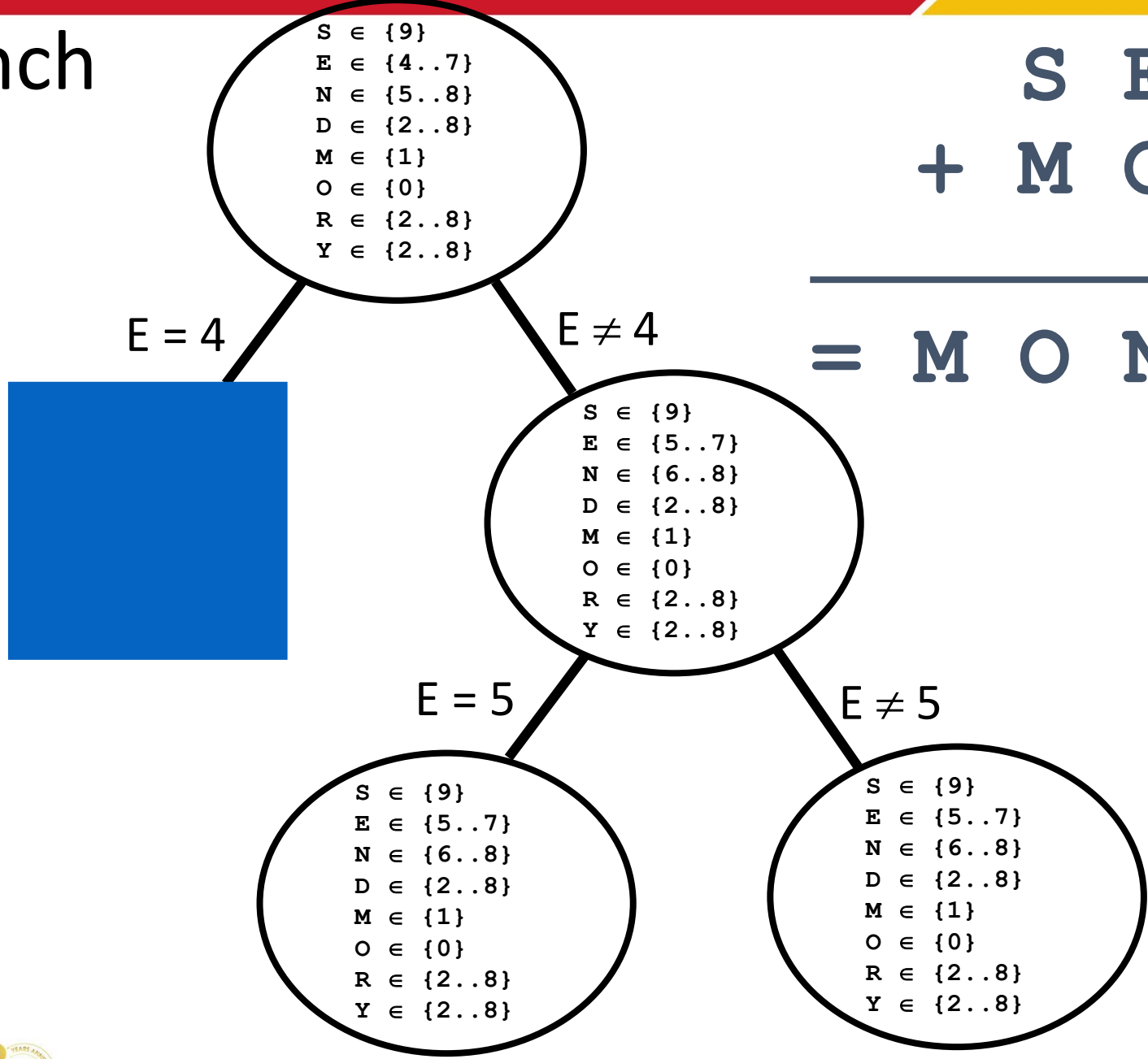
$S \neq 0$

$M \neq 0$

S...Y all different

$$\begin{aligned} & 1000*S + 100*E + 10*N + D \\ & + 1000*M + 100*O + 10*R + E \\ & = 10000*M + 1000*O + 100*N + 10*E + Y \end{aligned}$$

Branch



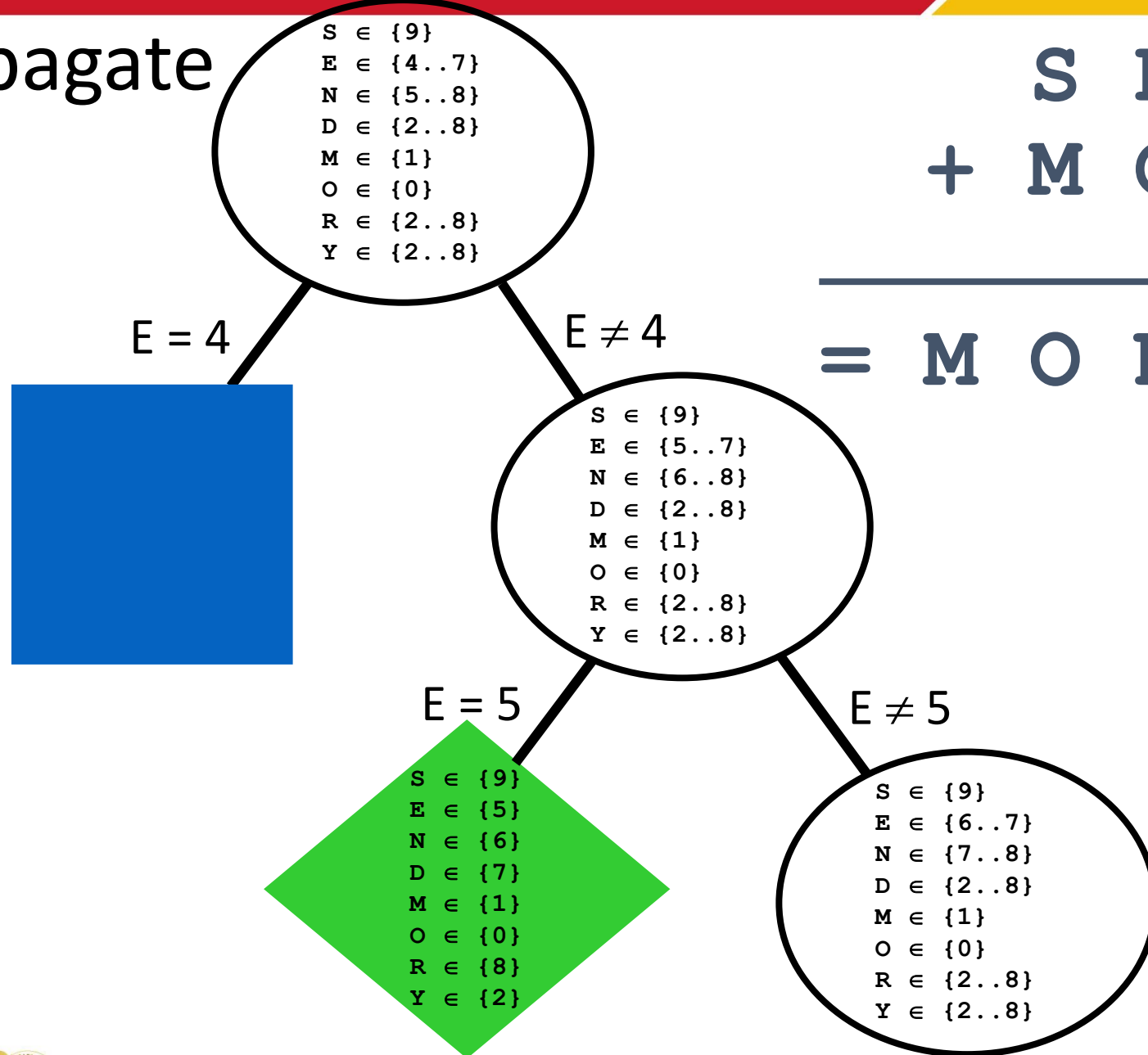
S E N D
+ M O R E

= M O N E Y

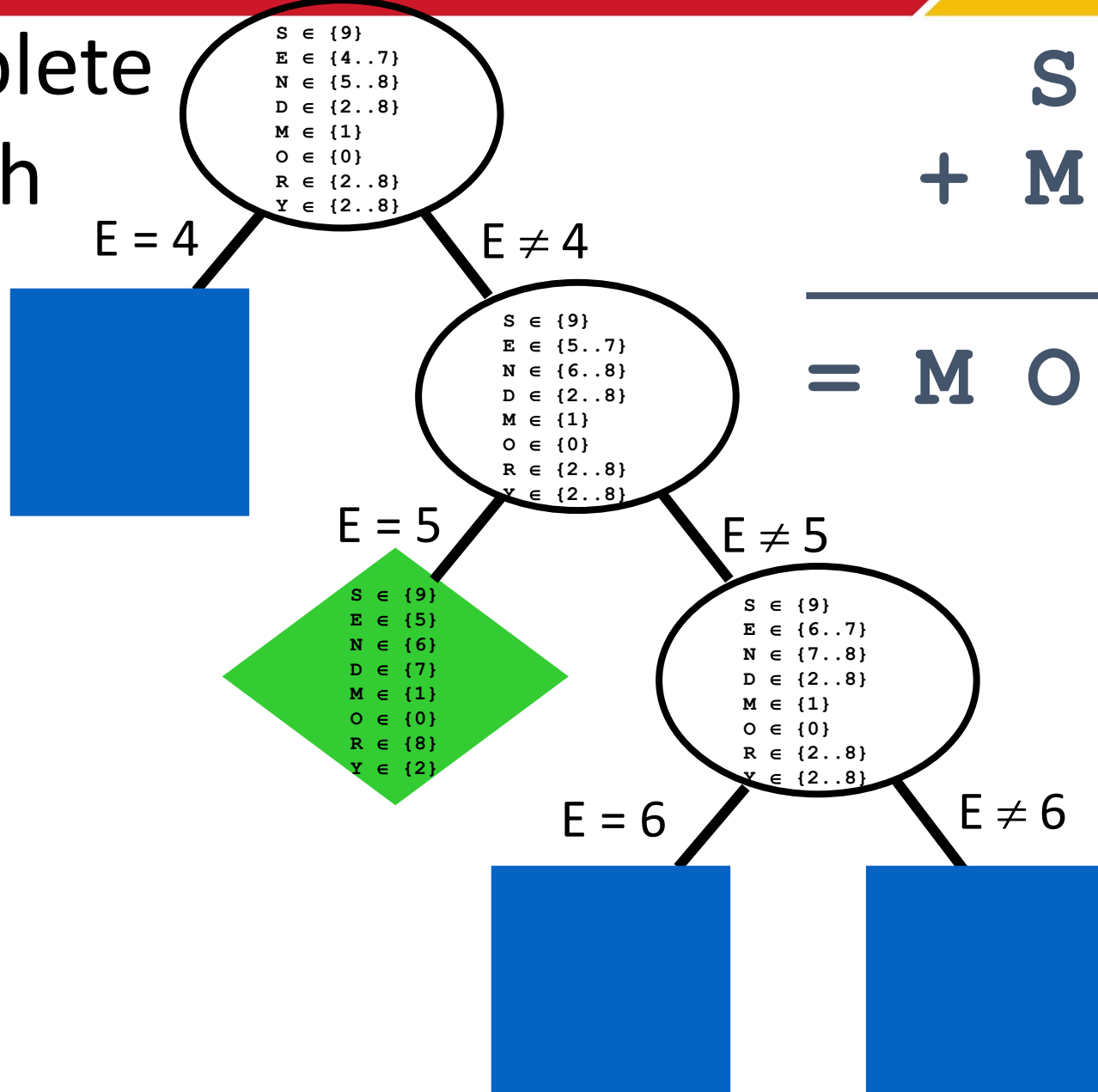
Propagate

S E N D
+ M O R E

= M O N E Y



Complete Search Tree



S E N D
+ M O R E

= M O N E Y

Constraint Programming Systems

Role: support elements of constraint programming

- Provide propagation algorithms for constraints
 - all different (e.g. wait for fixing)
 - summation (e.g. interval consistency)
- Allow choice of branching algorithm (e.g. first-fail)
- Allow choice of exploration algorithm (e.g. depth-first search)

The Art of Constraint Programming

- Choose model
- Choose propagation algorithms
- Choose branching algorithm
- Choose exploration algorithm

Nội dung

- Tổng quan phương pháp quy hoạch ràng buộc
- **Thành phần trong quy hoạch ràng buộc**
- Thu hẹp không gian tìm kiếm
- Phân nhánh và tìm kiếm quay lui
- Ví dụ minh họa bài toán N-queens
- Bài toán Sudoku
- Bài toán phân bổ môn học

Constraint Problems

A finite domain constraint problem consists of:

- number of variables: n
- constraints: $\mathbf{c}_1, \dots, \mathbf{c}_m \subseteq \mathbf{Z}^n$

The problem is to find

$\mathbf{a} = (\mathbf{v}_1, \dots, \mathbf{v}_n) \in \mathbf{Z}^n$ such that

$\mathbf{a} \in \mathbf{c}_i$, for all $1 \leq i \leq m$

Constraints

Constraint programming separates constraints into

- basic constraints: complete constraint solving
- non-basic constraints: propagation (incomplete); search needed

Basic Constraints and Propagators

`all_different(S,E,N,D,
M,O,R,Y)`

$$\begin{aligned} & 1000*S + 100*E + 10*N + D \\ & + 1000*M + 100*O + 10*R + E \\ = & 10000*M + 1000*O + 100*N + 10*E + Y \end{aligned}$$

$S \in \{1..9\}$
 $E \in \{0..9\}$
 $N \in \{0..9\}$
 $D \in \{0..9\}$
 $M \in \{1..9\}$
 $O \in \{0..9\}$
 $R \in \{0..9\}$
 $Y \in \{0..9\}$

Basic Constraints and Propagators

`all different(S,E,N,D,
M,O,R,Y)`

$$\begin{aligned} & 1000*S + 100*E + 10*N + D \\ & + 1000*M + 100*O + 10*R + E \\ = & 10000*M + 1000*O + 100*N + 10*E + Y \end{aligned}$$

$S \in \{1..9\}$
 $E \in \{0..9\}$
 $N \in \{0..9\}$
 $D \in \{0..9\}$
 $M \in \{1\}$
 $O \in \{0..9\}$
 $R \in \{0..9\}$
 $Y \in \{0..9\}$

Basic Constraints and Propagators

`all different(S,E,N,D,
M,O,R,Y)`

$$\begin{aligned} & 1000*S + 100*E + 10*N + D \\ & + 1000*M + 100*O + 10*R + E \\ = & 10000*M + 1000*O + 100*N + 10*E + Y \end{aligned}$$

$S \in \{2..9\}$
 $E \in \{0,2..9\}$
 $N \in \{0,2..9\}$
 $D \in \{0,2..9\}$
 $M \in \{1\}$
 $O \in \{0,2..9\}$
 $R \in \{0,2..9\}$
 $Y \in \{0,2..9\}$

Basic Constraints and Propagators

`all different(S,E,N,D,
M,O,R,Y)`

$$\begin{aligned} & 1000*S + 100*E + 10*N + D \\ & + 1000*M + 100*O + 10*R + E \\ = & 10000*M + 1000*O + 100*N + 10*E + Y \end{aligned}$$

$S \in \{2..9\}$
 $E \in \{0,2..9\}$
 $N \in \{0,2..9\}$
 $D \in \{0,2..9\}$
 $M \in \{1\}$
 $O \in \{0\}$
 $R \in \{0,2..9\}$
 $Y \in \{0,2..9\}$

and so on and so on

Basic Constraints and Propagators

`all different(S,E,N,D,
M,O,R,Y)`

$$\begin{aligned} & 1000*S + 100*E + 10*N + D \\ & + 1000*M + 100*O + 10*R + E \\ = & 10000*M + 1000*O + 100*N + 10*E + Y \end{aligned}$$

$S \in \{9\}$
 $E \in \{5..7\}$
 $N \in \{6..8\}$
 $D \in \{2..8\}$
 $M \in \{1\}$
 $O \in \{0\}$
 $R \in \{2..8\}$
 $Y \in \{2..8\}$

Issues in Propagation

- Completeness: What behavior can be expected from propagation?
- Efficiency: How much computational resources does propagation consume?

Completeness of Propagation

- Given: Basic constraint C and propagator P .
- Propagation is complete, if for every variable \mathbf{x} and every value \mathbf{v} in the domain of \mathbf{x} , there is an assignment in which $\mathbf{x}=\mathbf{v}$ that satisfies C and P .
- Complete propagation is also called *domain-consistency* or *arc-consistency*.

Completeness of Propagation

- General arithmetic constraints are undecidable (Hilbert's Tenth Problem).
- Propagation cannot always exhibit all inconsistencies.
- Example:

$$c_1: \quad n > 2$$

$$c_2: \quad a^n + b^n = c^n$$

Example: Complete All Different

- C: $w \in \{1, 2, 3, 4\}$
 $x \in \{2, 3, 4\}$
 $y \in \{2, 3, 4\}$
 $z \in \{2, 3, 4\}$
- P: `all_different(w, x, y, z)`

Example: Complete All Different

- C: $w \in \{1, 2, \underline{3}, 4\}$
 $x \in \{2, 3, 4\}$
 $y \in \{2, 3, 4\}$
 $z \in \{2, 3, 4\}$
- P: `all_different(w, x, y, z)`
- Most efficient known algorithm: $O(|X|^2 d_{\max}^2)$
Regin [1994], using graph matching

Basic Constraints vs. Propagators

- Basic constraints
 - are conjunctions of constraints of the form $\mathbf{x} \in \mathbf{S}$, where \mathbf{S} is a finite set of integers
 - Enjoy complete constraint solving
- Propagators
 - can be arbitrarily expressive (arithmetic, symbolic)
 - implementation typically fast but incomplete

Propagation vs Branching

Obvious trade-off:

Complex propagation algorithms result in fewer, but more expensive nodes in the search tree.

Example: **MONEY** with

alldiff and **sum**:
only test fixed
assignment

alldiff: wait
for fixed variables
sum: interval cons.

alldiff and **sum**:
domain
consistency

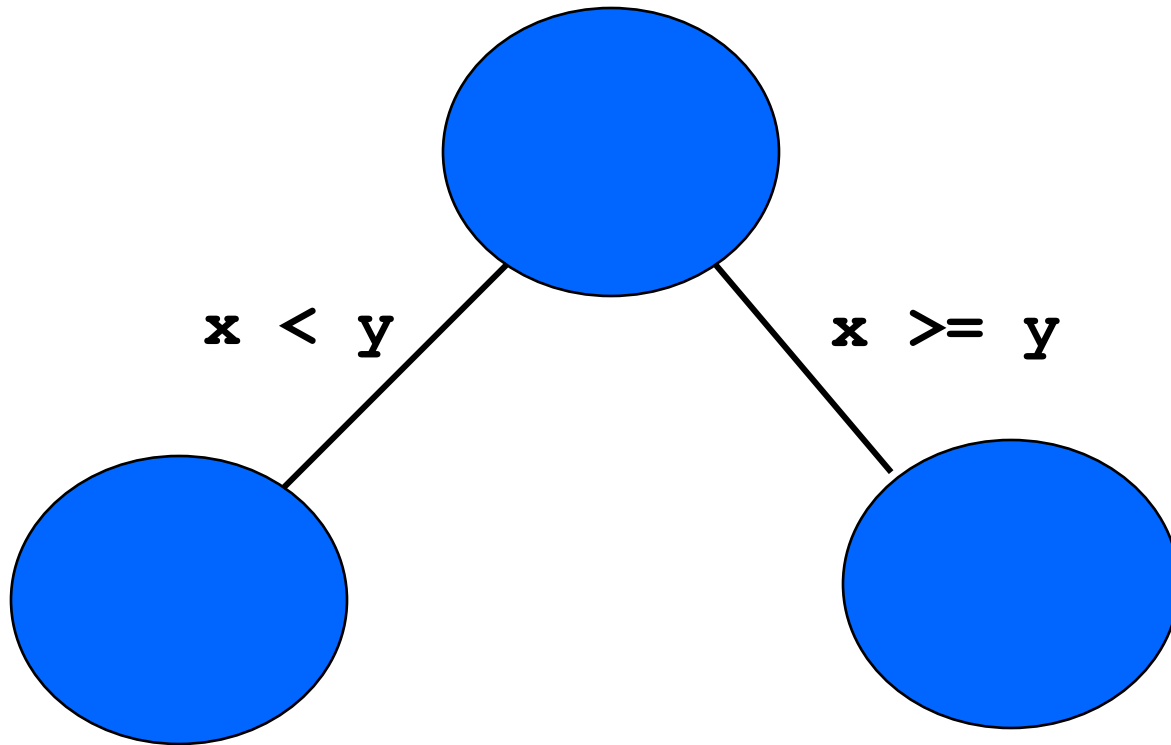
Branching Algorithms

Constraint programming systems provide

- libraries of predefined branching algorithms
- programming support for user-defined branching algorithms

Basic Choice Points

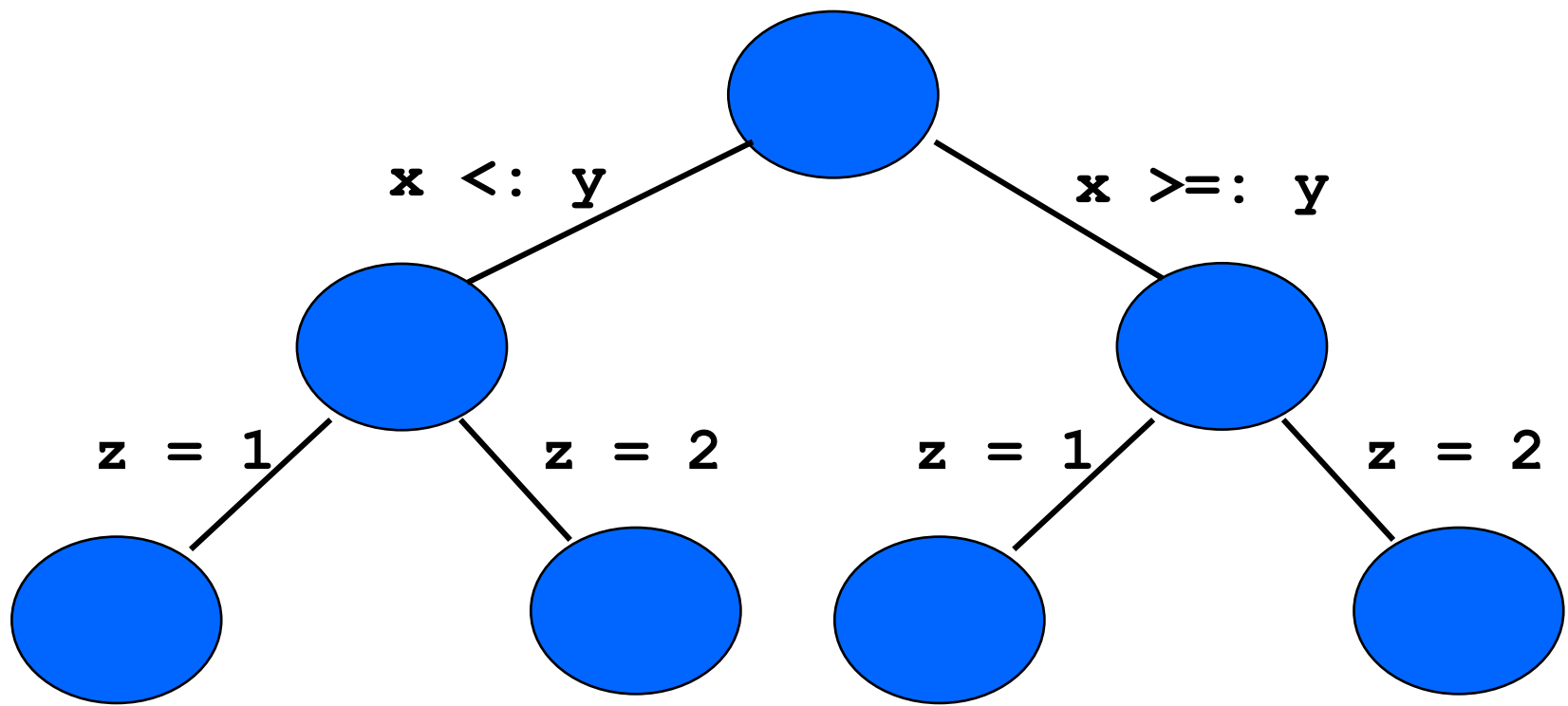
choice $x <: y$ [] $x >=: y$ **end**



Choice Point Sequences

choice $x <: y$ [] $x >=: y$ **end**

choice $z = 1$ [] $z = 2$ **end**



Examples of Branching Algorithms

- Enumeration: Choose variable, choose value
 - naive enumeration: choose variables and values in a fixed sequence
 - first-fail enumeration: choose a variable with minimal domain size
- Domain-splitting:
choice X <: Mid [] X >=: Mid **end**
- Task sequencing for scheduling (later)

Order of Exploration

- Depth-first search
- Iterative Deepening

Optimization

- Branch-and-bound
- Restart optimization

Thu hẹp không gian tìm kiếm

- Định nghĩa: Bài toán CSP = (X, D, C) , trong đó:
 - $X = \{X_1, \dots, X_N\}$ – tập các biến
 - $D = \{D(X_1), \dots, D(X_N)\}$ – tập miền giá trị của các biến
 - $C = \{C_1, \dots, C_K\}$ – tập các ràng buộc
 - Ký hiệu $X(c)$ – tập các biến tham gia vào ràng buộc c

Thu hẹp không gian tìm kiếm

- Domain consistency (DC)
 - Cho bài toán thỏa mãn ràng buộc $CSP = (X, D, C)$, một ràng buộc $c \in C$ được gọi là domain consistent nếu với mỗi biến $X_i \in X(c)$, mỗi giá trị $v \in D(X_i)$, tồn tại bộ giá trị cho các biến $\in X(c) \setminus \{X_i\}$ sao cho ràng buộc c được thỏa mãn
 - Bài toán CSP được gọi là domain consistent nếu c là domain consistent với mọi ràng buộc $c \in C$
- Thuật toán DC là thuật toán nhằm loại bỏ các giá trị dư thừa khỏi miền giá trị của các biến để đưa bài toán CSP ban đầu về bài toán CSP domain consistent tương đương

Thu hẹp không gian tìm kiếm

- Ví dụ: CSP = (X, D, C) trong đó:

- $X = \{X_1, X_2, X_3, X_4\}$
- $D(X_1) = \{1, 2, 3, 4\}$, $D(X_2) = \{1, 2, 3, 4, 5, 6, 7\}$, $D(X_3) = \{2, 3, 4, 5\}$, $D(X_4) = \{1, 2, 3, 4, 5, 6\}$
- $C = \{c_1, c_2, c_3\}$ 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 này là domain consistent

- Khi phân nhánh, xét $X_1 = 1$, thuật toán DC sẽ đưa CSP đã cho về CSP¹ tương đương các là domain consistent với miền giá trị được thu hẹp như sau : $D^1(X_1) = \{1\}$, $D^1(X_2) = \{4, 5, 6, 7\}$, $D^1(X_3) = \{2, 3, 4\}$, $D^1(X_4) = \{1, 2, 3, 4, 5\}$

Thu hẹp không gian tìm kiếm

- Một bài toán CSP là domain consistent chưa đảm bảo luôn có lời giải,
 - Ví dụ xét CSP sau:
 - $X = \{X_1, X_2, X_3\}$
 - $D(X_1) = D(X_2) = D(X_3) = \{0, 1\}$
 - $c_1 \equiv X_1 \neq X_2$, $c_2 \equiv X_1 \neq X_3$, $c_3 \equiv X_2 \neq X_3$
- Rõ ràng CSP này là domain consistent nhưng lại không có lời giải chấp nhận được (lời giải thỏa mãn ràng buộc)

Thu hẹp không gian tìm kiếm

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;

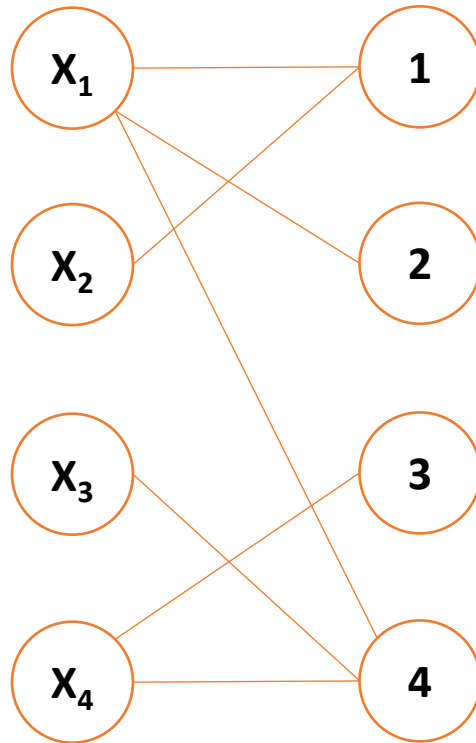
}

Thu hẹp không gian tìm kiếm

- Một số ràng buộc, ví dụ ràng buộc nhị phân (ràng buộc giữa 2 biến) → thuật toán DC hiệu quả
- Ràng buộc AllDifferent(X_1, X_2, \dots, X_N), thuật toán DC dựa trên thuật toán hiệu quả cặp ghép cực đại (Max-Matching)
 - Tập đỉnh bên phải là các biến, tập đỉnh bên trái là các giá trị
 - Với mỗi cạnh (X_i, v) , (với $v \in D(X_i)$), nếu không tồn tại phương án cặp ghép kích thước N trong đó (X_i, v) là một thành phần của phương án thì có thể loại bỏ v khỏi $D(X_i)$

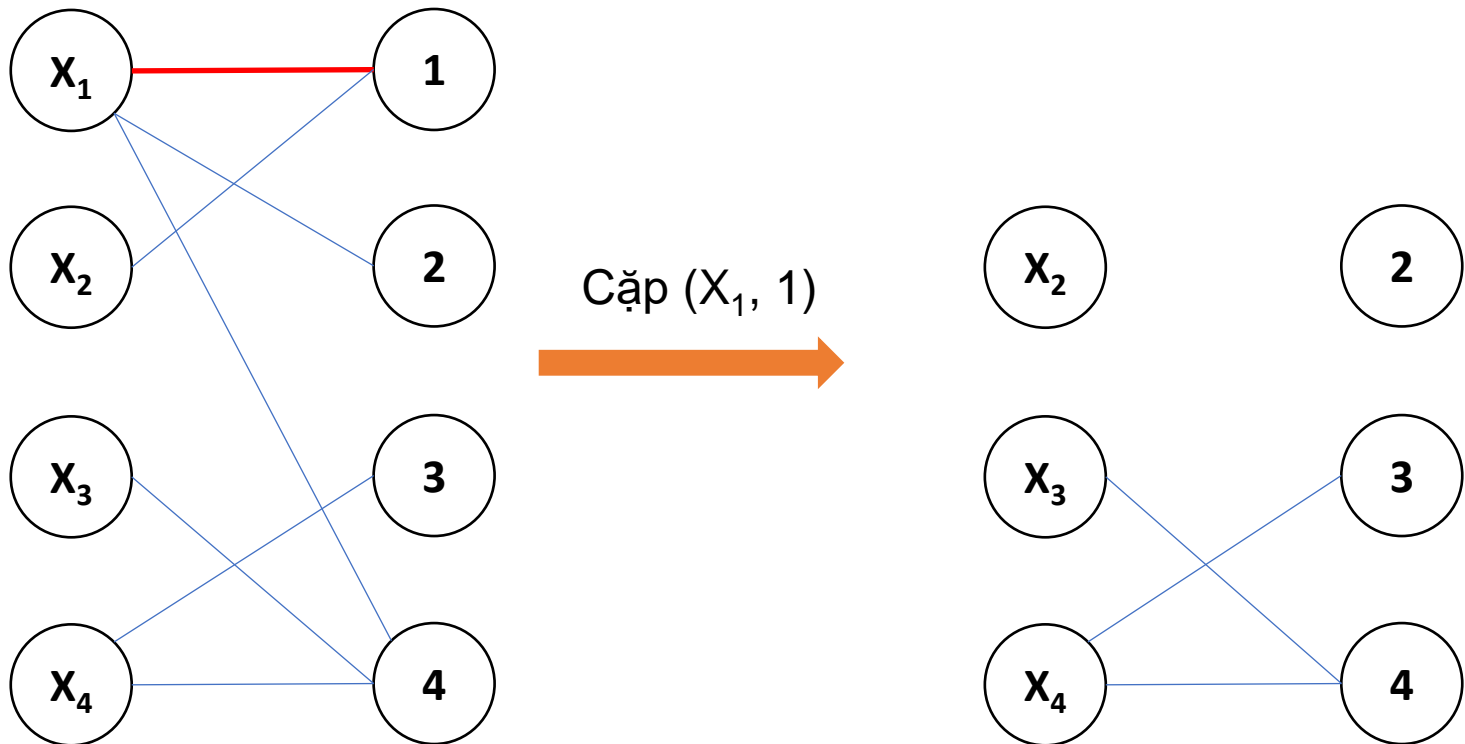
Ràng buộc AllDifferent

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



Ràng buộc AllDifferent

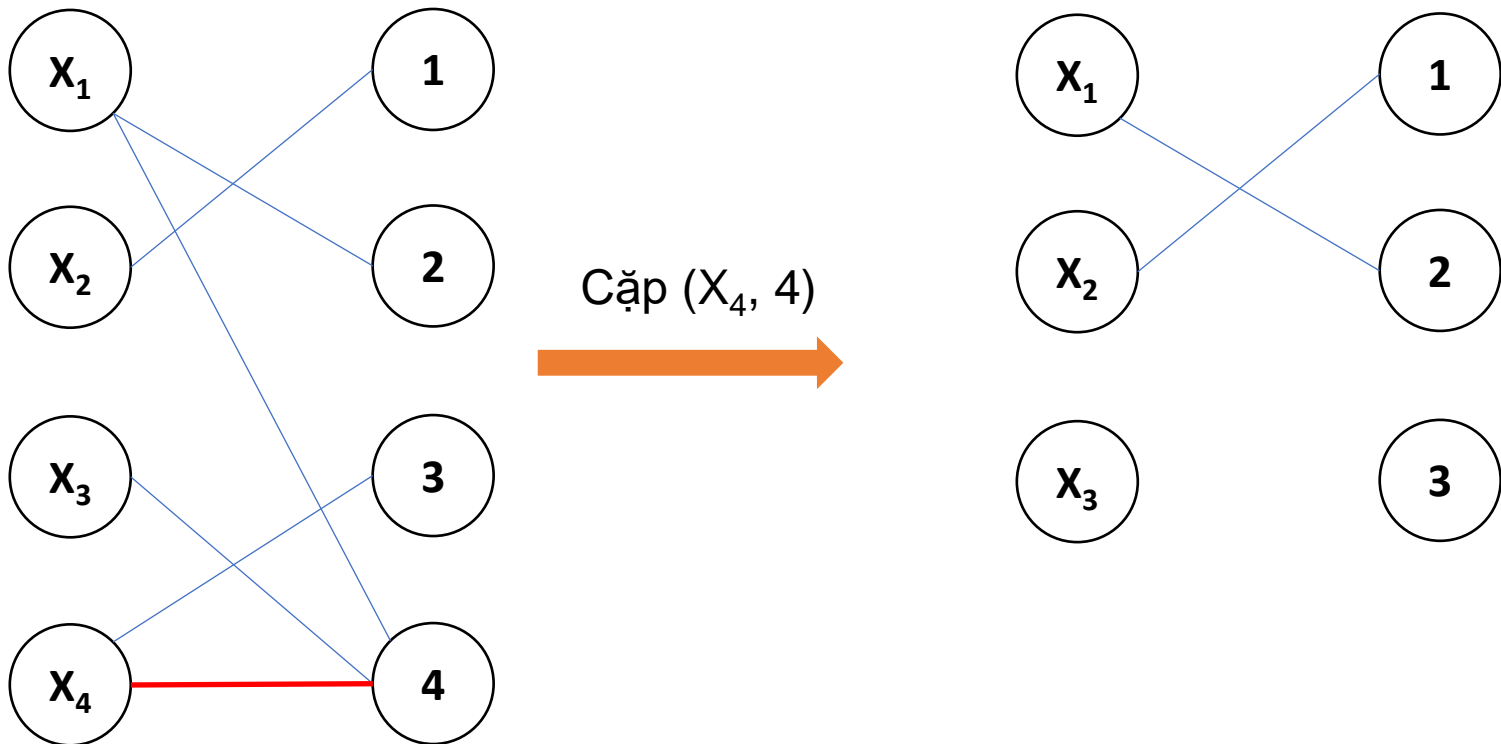
- $X = \{X_1, X_2, X_3, X_4\}$
- $D(X_1) = \{1, 2, 4\}$, $D(X_2) = \{1\}$, $D(X_3) = \{4\}$, $D(X_4) = \{3, 4\}$



Không tồn tại cặp ghép kích thước 3 \rightarrow loại bỏ được 1 khỏi miền giá trị của X_1

Ràng buộc AllDifferent

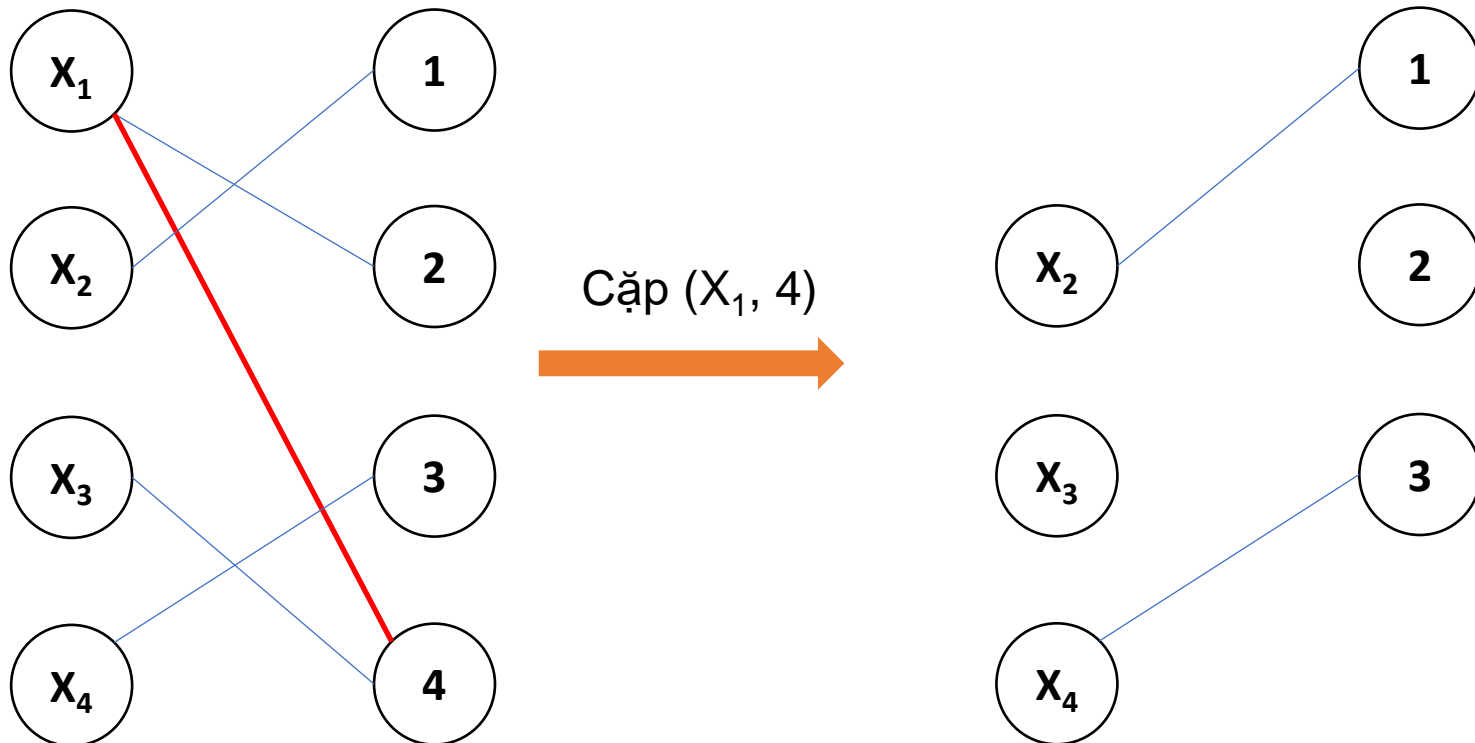
- $X = \{X_1, X_2, X_3, X_4\}$
- $D(X_1) = \{2,4\}$, $D(X_2) = \{1\}$, $D(X_3) = \{4\}$, $D(X_4) = \{3,4\}$



Không tồn tại cặp ghép kích thước 3 \rightarrow loại
bỏ được 4 khỏi miền giá trị của X_4

Ràng buộc AllDifferent

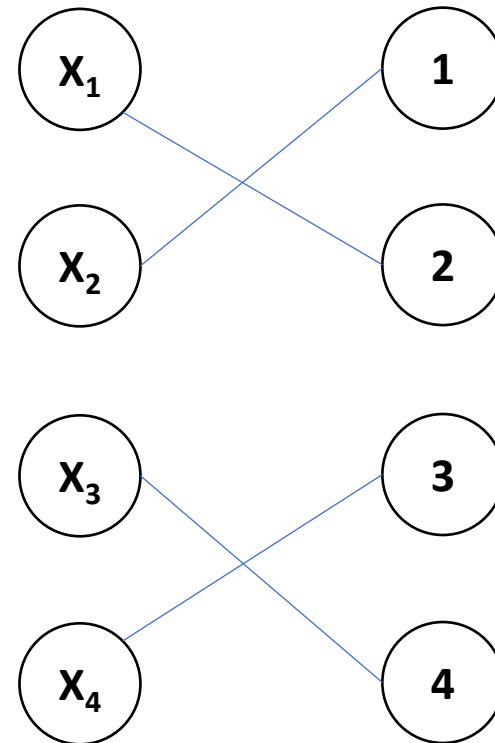
- $X = \{X_1, X_2, X_3, X_4\}$
- $D(X_1) = \{2, 4\}$, $D(X_2) = \{1\}$, $D(X_3) = \{4\}$, $D(X_4) = \{3\}$



Không tồn tại cặp ghép kích thước 3 \rightarrow loại
bỏ được 4 khỏi miền giá trị của X_1

Ràng buộc AllDifferent

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



Phương án chấp nhận được

Phân nhánh và tìm kiếm quay lui

- Việc tả không gian tìm kiếm (Propagation) không đủ để tìm ra lời giải tối ưu thỏa mãn ràng buộc
- Cần thiết phải kết hợp giữa tả không gian tìm kiếm với phân nhánh và tìm kiếm quay lui

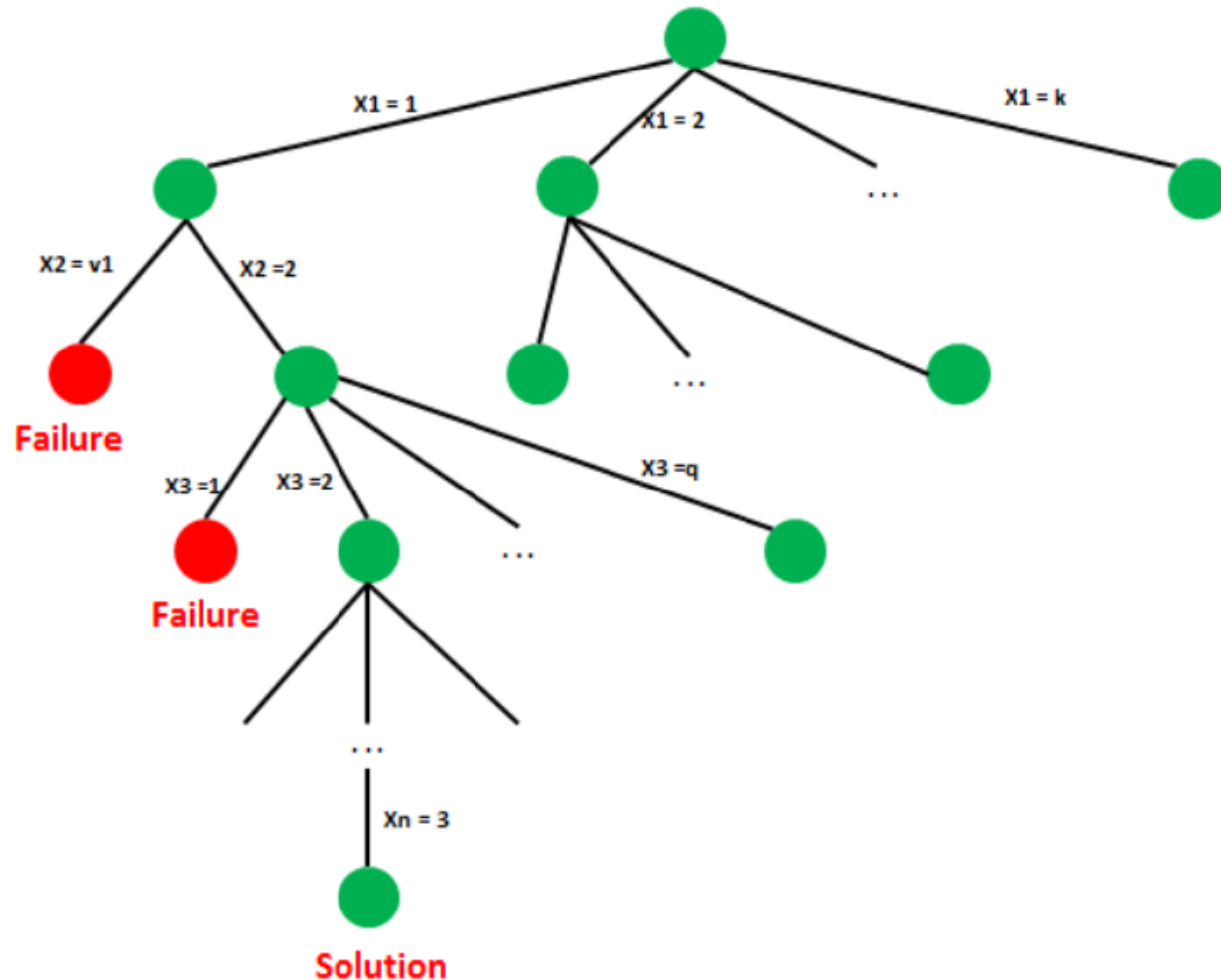
Phân nhánh và tìm kiếm quay lui

- Việc tĩa không gian tìm kiếm (Propagation) không đủ để tìm ra lời giải tối ưu thỏa mãn ràng buộc
- Cần thiết phải kết hợp giữa tĩa không gian tìm kiếm với phân nhánh và tìm kiếm quay lui
 - Phân ra bài toán CSP P_0 ban đầu thành các CSP P_1, \dots, P_M
 - Tập các lời giải của P_0 bằng hợp của tập các lời giải của P_1, \dots, P_M
 - Miền giá trị mỗi biến trong P_1, \dots, P_M không lớn hơn miền giá trị P_0

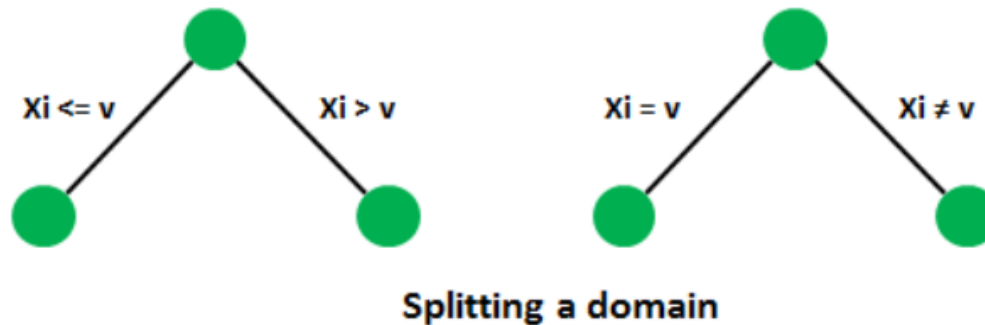
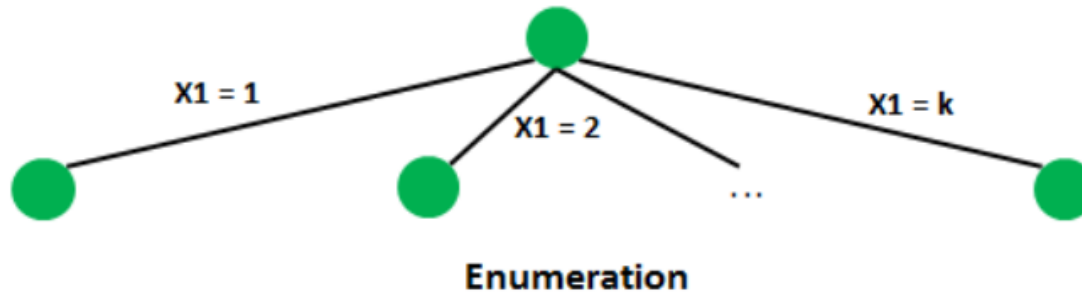
Phân nhánh và tìm kiếm quay lui

- Việc tả không gian tìm kiếm (Propagation) không đủ để tìm ra lời giải tối ưu thỏa mãn ràng buộc
- Cần thiết phải kết hợp giữa tả không gian tìm kiếm với phân nhánh và tìm kiếm quay lui
 - Phân ra bài toán CSP P_0 ban đầu thành các CSP P_1, \dots, P_M
 - Tập các lời giải của P_0 bằng hợp của tập các lời giải của P_1, \dots, P_M
 - Miền giá trị mỗi biến trong P_1, \dots, P_M không lớn hơn miền giá trị P_0
 - Cây tìm kiếm (Search Tree)
 - Nút gốc là CSP P_0 ban đầu
 - Mỗi nút của cây là 1 CSP
 - Nếu P_1, \dots, P_M là các nút con của P_0 thì tập các lời giải của P_0 sẽ bằng với hợp của tập các lời giải của P_1, \dots, P_M
 - Nút lá
 - Một lời giải thỏa mãn ràng buộc
 - Failure (tồn tại biến của miền giá trị rỗng)

Phân nhánh và tìm kiếm quay lui



Phân nhánh và tìm kiếm quay lui



Phân nhánh và tìm kiếm quay lui

- Một số chiến lược tìm kiếm
 - Chọn biến
 - **dom** heuristic: chọn biến có miền giá trị nhỏ nhất
 - **deg** heuristic: chọn biến tham gia vào nhiều ràng buộc nhất
 - **dom+deg** heuristic: áp dụng dom trước, sau đó áp dụng deg khi có nhiều biến có cùng kích thước miền giá trị nhỏ nhất
 - **dom/deg**: chọn biến có tỉ số dom/deg nhỏ nhất (kích thước miền giá trị/số ràng buộc mà biến tham gia vào)
 - Chọn giá trị
 - Chọn giá trị theo thứ tự tăng dần
 - Chọn giá trị theo thứ tự giảm dần
 - Chọn giá trị ở giữa miền giá trị nhất

Một số thư viện

- Gecode: <https://www.gecode.org/>
- Minizinc: <https://www.minizinc.org/>
- Google OR-tools: <https://developers.google.com/optimization>
- CHOCO: <https://github.com/chocoteam/choco-solver>

Ví dụ

- Biến
 - $X = \{X_0, X_1, X_2, X_3, X_4\}$
- Miền giá trị
 - $X_0, X_1, X_2, X_3, X_4 \in \{1, 2, 3, 4, 5\}$
- Ràng buộc
 - $C_1: X_2 + 3 \neq X_1$
 - $C_2: X_3 \leq X_4$
 - $C_3: X_2 + X_3 = X_0 + 1$
 - $C_4: X_4 \leq 3$
 - $C_5: X_1 + X_4 = 7$
 - $C_6: X_2 = 1 \Rightarrow X_4 \neq 2$

Ví dụ

```
from ortools.sat.python import cp_model

model = cp_model.CpModel()

x0 = model.NewIntVar(1, 5, 'x0')
x1 = model.NewIntVar(1, 5, 'x1')
x2 = model.NewIntVar(1, 5, 'x2')
x3 = model.NewIntVar(1, 5, 'x3')
x4 = model.NewIntVar(1, 5, 'x4')

model.Add(x2 + 3 != x1)
model.Add(x3 <= x4)
model.Add(x2 + x3 == x0 + 1)
model.Add(x4 <= 3)
model.Add(x1 + x4 == 7)

b = model.NewBoolVar('b')
model.Add(x2 == 1).OnlyEnforceIf(b)
model.Add(x2 != 1).OnlyEnforceIf(b.Not())
model.Add(x4 != 2).OnlyEnforceIf(b)

solver = cp_model.CpSolver()
status = solver.Solve(model)

if status == cp_model.OPTIMAL:
    print('Maximum of objective function: %i' % solver.ObjectiveValue())
    print()
    print('x0 value: ', solver.Value(x0))
    print('x1 value: ', solver.Value(x1))
    print('x2 value: ', solver.Value(x2))
    print('x3 value: ', solver.Value(x3))
    print('x4 value: ', solver.Value(x4))
```

Ví dụ: CP + IS + FUN = TRUE

```
def CPisFunSat():  
    """Solve the CP+IS+FUN==TRUE cryptarithm."""  
    # Constraint programming engine  
    model = cp_model.CpModel()  
  
    base = 10  
  
    c = model.NewIntVar(1, base - 1, 'C')  
    p = model.NewIntVar(0, base - 1, 'P')  
    i = model.NewIntVar(1, base - 1, 'I')  
    s = model.NewIntVar(0, base - 1, 'S')  
    f = model.NewIntVar(1, base - 1, 'F')  
    u = model.NewIntVar(0, base - 1, 'U')  
    n = model.NewIntVar(0, base - 1, 'N')  
    t = model.NewIntVar(1, base - 1, 'T')  
    r = model.NewIntVar(0, base - 1, 'R')  
    e = model.NewIntVar(0, base - 1, 'E')  
  
    # We need to group variables in a list to use the constraint AllDifferent.  
    letters = [c, p, i, s, f, u, n, t, r, e]
```

Ví dụ: CP + IS + FUN = TRUE

```
# Verify that we have enough digits.
assert base >= len(letters)

# Define constraints.
model.AddAllDifferent(letters)

# CP + IS + FUN = TRUE
model.Add(c * base + p + i * base + s + f * base * base + u * base +
          n == t * base * base * base + r * base * base + u * base + e)

### Solve model.
solver = cp_model.CpSolver()
solution_printer = VarArraySolutionPrinter(letters)

# Enumerate all solutions.
solver.parameters.enumerate_all_solutions = True
# Solve.
status = solver.Solve(model, solution_printer)

print()
print('Statistics')
print(' - status           : %s' % solver.StatusName(status))
print(' - conflicts          : %i' % solver.NumConflicts())
print(' - branches           : %i' % solver.NumBranches())
print(' - wall time          : %f s' % solver.WallTime())
print(' - solutions found : %i' % solution_printer.solution_count())
```

Ví dụ: CP + IS + FUN = TRUE

```
class VarArraySolutionPrinter(cp_model.CpSolverSolutionCallback):
    """Print intermediate solutions."""

    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(self):
        return self.__solution_count
```

Bài toán N-queen

```
def main(board_size):
    model = cp_model.CpModel()
    # Creates the variables.
    # The array index is the column, and the value is the row.
    queens = [model.NewIntVar(0, board_size - 1, 'x%i' % i)
               for i in range(board_size)]
    # Creates the constraints.
    # The following sets the constraint that all queens are in different rows.
    model.AddAllDifferent(queens)

    # Note: all queens must be in different columns because the indices
    # of queens are all different.
```

Bài toán N-queen

```
# The following sets the constraint that no two queens can be on
# the same diagonal.
for i in range(board_size):
    # Note: is not used in the inner loop.
    diag1 = []
    diag2 = []
    for j in range(board_size):
        # Create variable array for queens(j) + j.
        q1 = model.NewIntVar(0, 2 * board_size, 'diag1_%i' % i)
        diag1.append(q1)
        model.Add(q1 == queens[j] + j)
        # Create variable array for queens(j) - j.
        q2 = model.NewIntVar(-board_size, board_size, 'diag2_%i' % i)
        diag2.append(q2)
        model.Add(q2 == queens[j] - j)
    model.AddAllDifferent(diag1)
    model.AddAllDifferent(diag2)
```

Bài toán N-queen

```
# The following sets the constraint that no two queens can be on
# the same diagonal.
for i in range(board_size):
    # Note: is not used in the inner loop.
    diag1 = []
    diag2 = []
    for j in range(board_size):
        # Create variable array for queens(j) + j.
        q1 = model.NewIntVar(0, 2 * board_size, 'diag1_%i' % i)
        diag1.append(q1)
        model.Add(q1 == queens[j] + j)
        # Create variable array for queens(j) - j.
        q2 = model.NewIntVar(-board_size, board_size, 'diag2_%i' % i)
        diag2.append(q2)
        model.Add(q2 == queens[j] - j)
    model.AddAllDifferent(diag1)
    model.AddAllDifferent(diag2)
```

Bài toán N-queen

```
### Solve model.
solver = cp_model.CpSolver()
solution_printer = SolutionPrinter(queens)
status = solver.SearchForAllSolutions(model, solution_printer)
print()
print('Solutions found : %i' % solution_printer.SolutionCount())

class SolutionPrinter(cp_model.CpSolverSolutionCallback):
    """Print intermediate solutions."""

    def __init__(self, variables):
        cp_model.CpSolverSolutionCallback.__init__(self)
        self.__variables = variables
        self.__solution_count = 0

    def OnSolutionCallback(self):
        self.__solution_count += 1
        for v in self.__variables:
            print('%s = %i' % (v, self.Value(v)), end = ' ')
        print()

    def SolutionCount(self):
        return self.__solution_count
```


Bài toán Phân bổ môn học

- Có N môn học $1, 2, \dots, N$ cần được phân bổ vào P học kỳ $1, 2, \dots, P$
- Mỗi môn học i có số tín chỉ là $credit(i)$
- $L = \{(i, j)\}$: tập các cặp môn học (i, j) trong điều kiện tiên quyết (môn i phải được xếp và học kỳ trước học kỳ của môn j)
- Cho trước các hằng số $\alpha, \beta, \lambda, \gamma$. Hãy tìm cách xếp N môn học vào P học kỳ sao cho
 - Tổng số môn học trong mỗi học kỳ phải lớn hơn hoặc bằng α và nhỏ hơn hoặc bằng β
 - Tổng số tín chỉ các môn học trong mỗi học kỳ phải lớn hơn hoặc bằng λ và nhỏ hơn hoặc bằng γ

Bài toán Phân bổ môn học


Môn	1	2	3	4	5	6	7	8	9	10	11	12
Số tín chỉ	2	1	2	1	3	2	1	3	2	3	1	3

$3 \leq \text{Số môn học mỗi học kỳ} \leq 3$

$5 \leq \text{Số tín chỉ các môn học mỗi học kỳ} \leq 7$

**Phương án
phân bổ**

Học kỳ	1	2	3	4
Danh sách môn	2, 5, 3	1, 6,10	4,7,8	9,11,12



2	1
6	9
5	6
5	8
4	11
6	12
2	7
3	10
5	7
8	11
4	12

Bài toán Phân bổ môn học

- Biến
 - $X[p,i] = 1$: môn i được phân vào học kỳ p
 - $D(X[p,i]) = \{0,1\}$
- Ràng buộc
 - $X[q,i] = 1 \rightarrow X[p,j] = 0, (i,j) \in L, 1 \leq p \leq q \leq P$
 - $\sum_{p=1}^P X[p,i] = 1$, với mọi $i = 1,2,\dots,N$
 - $\alpha \leq \sum_{i=1}^N X[p,i] \leq \beta$, với mọi $p = 1,2,\dots,P$
 - $\lambda \leq \sum_{i=1}^N X[p,i] \text{credit}(i) \leq \gamma$, với mọi $p = 1,2,\dots,P$

Bài toán Phân bổ môn học

```
N = 12
P = 4
credits = [2, 1, 2, 1, 3, 2, 1, 3, 2, 3, 1, 3]

orderCourseLst = [(0, 1), (1, 2)]

alpha = 3
beta = 3
lamb = 5
gamma = 7

model = cp_model.CpModel()

# Khai báo biến x[i,j] = 1 nếu môn học j được phân vào kỳ p
x = []
for i in range(P):
    t = []
    for j in range(N):
        t.append(model.NewIntVar(0, 1, 'x[' + str(i) + "," + str(j) + "]"))
    x.append(t)
```

Bài toán Phân bổ môn học

```
#Ràng buộc: Mỗi môn học chỉ được phân vào duy nhất 1 kỳ
for j in range(N):
    model.Add(sum(x[i][j] for i in range(P)) == 1)

#Ràng buộc: Số lượng môn học trong một kỳ phải nằm trong [alpha, beta]
for i in range(P):
    model.Add(sum(x[i][j] for j in range(N)) >= alpha)
    model.Add(sum(x[i][j] for j in range(N)) <= beta)

#Ràng buộc: Số tín chỉ trong một kỳ phải nằm trong [lambda, gamma]
for i in range(P):
    model.Add(sum(x[i][j]*credits[j] for j in range(N)) >= lamb)
    model.Add(sum(x[i][j]*credits[j] for j in range(N)) <= gamma)
```

Bài toán Phân bổ môn học

```
#Ràng buộc: Thứ tự các môn học
for item in orderCourseLst:
    i = item[0]
    j = item[1]

    for q in range(P):
        for p in range(q+1):

            b = model.NewBoolVar('b')
            model.Add(x[q][i] == 1).OnlyEnforceIf(b)
            model.Add(x[q][i] != 1).OnlyEnforceIf(b.Not())
            model.Add(x[p][j] == 0).OnlyEnforceIf(b)
```

Bài toán Phân bổ môn học

```
### Solve model.
solver = cp_model.CpSolver()
status = solver.Solve(model)

if status == cp_model.OPTIMAL or status == cp_model.FEASIBLE:
    print(f'Total cost = {solver.ObjectiveValue()}')
    print()
    for i in range(P):
        for j in range(N):
            if solver.BooleanValue(x[i][j]):
                print(
                    f'Môn học {j} được phân cho kỳ {i}')
else:
    print('No solution found.')
```



25 YEARS ANNIVERSARY
SOICT

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



soict.hust.edu.vn/



fb.com/groups/soict

