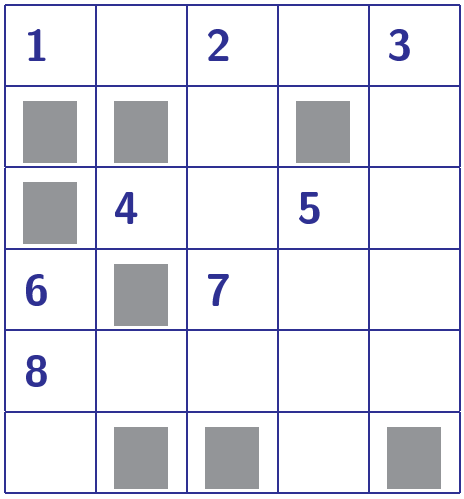
人工知能特論： GSAT　　第 3 回

4/9 講義の概要と予備知識. 充足可能性 (SAT)
4/16 David-Putnum 法
4/23 GSAT
4/30 プログラム演習 (休講) 課題 (1)： Davis-Putnum 法 及び GSAT
5/7 局所整合性 (Local Consistency)
5/14 後ろ戻り (Backtracking) 探索アルゴリズム
5/21 ソフトウェアエージェント (角助教授)
5/28 知識表現 (角助教授)
6/4 プラニング (角助教授)
6/11 マルチエージェント (角助教授)
　　課題 (2)： ソフトウェアエージェントシステム構築
6/18 創立記念日
6/25 休講
7/2 問題解決システム・推論エンジン・真偽維持システム
7/9 予備日
　　課題 (3)： AI システムの応用についてのレポート (仮)

簡単なクロスワードパズル



AFT
ALE
EEL
HEEL
HIKE
HOSES
KEEL
KNOT
LASER
LEE
LINE
SAILS
SHEET
STEER
TIE

制約充足問題 (Constraint Satisfaction Problems, CSPs)

2 項制約充足問題

(binary constraint satisfaction problem) とは

- 変数の集合 $\{x_1, x_2, \ldots, x_n\}$

  各変数の領域 (domain) は有限： $D_1, D_2, \ldots, D_n$

  - $D = D_1 \cup D_2 \cup \cdots \cup D_n$
  - d： 最大領域のサイズ
- 2 項制約 (binary constaints) e 個の集合 $\{C_{ij}\}$
  - C_{ij} は変数 x_i と x_j の許容された値の組合せの集合を指定する制約.
  - $C_{ij}(u, v) = C_{ji}(v, u)$ とする.

制約グラフ／ネットワーク (Constraint graph/network)

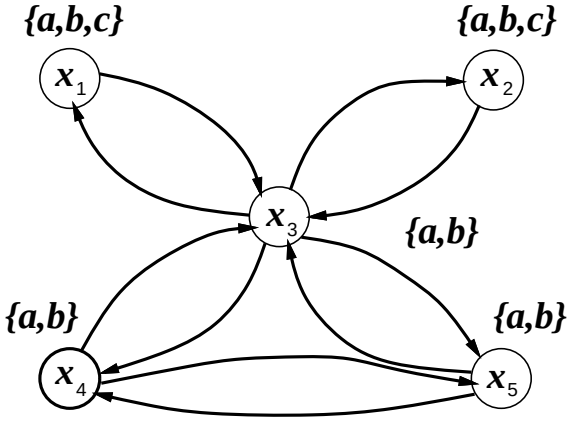
n 個の と e 個の (辺) を持つ有向グラフ

- 各変数は $(node)$ で表現
- 各制約 C_{ij} はノード x_i からノード x_j への (arc)

変数 $\{x_1, x_2, x_3, x_4, x_5\}$

制約

- $C_{13} = \{(a, b), (a, c), (b, c)\}$
- $C_{23} = \{(a, b), (a, c), (b, c)\}$
- $C_{34} = \{(a, b), (b, a)\}$
- $C_{35} = \{(a, b), (b, b)\}$
- $C_{45} = \{(a, b), (b, a)\}$



後戻り型探索 (Backtrack Search)

```
procedure  backtrack-search(n)
    consistent = true
    i =          ()
    loop
        if consistent then (i, consistent) = label(i)
        else (i, consistent) = unlabel(i)
        if i > n  then return "solution found"
        else if i = 0  then return "no solution"
    endloop
end  backtrack-search
```


時間的後戻り (Chronological Backtracking)：

```
function  initialize()
    for i = 1 to n
        CD_i = D_i          /* 現在の領域で初期化 */
    endfor
    return 1                /* 最初の変数を返す */
end  initialize
```

時間的後戻り (Chronological Backtracking)：*label*

```
procedure  bt-label(i)
    for each v_k ∈ CD_i  do
    Set x_i = v_k  and  consistent = true
    for j  from 1  to  i−1  do      /* 対既割当変数チェック */
        if ¬ C_ij(x_i, x_j) then
            Remove v_k from CD_i
                    and set consistent = false
            Unassign x_i  and  break  inner loop
        endif
    if consistent  then return (i + 1, true)
    endfor
    return (i, false)
end  bt-label
```


時間的後戻り (Chronological Backtracking)：*unlabel*

```
procedure  bt-unlabel(i)
   h = i − 1            /* 直前に割当てた変数に戻る */
   CD_i = D_i
    Remove current value assigned to x_h from CD_h
    Unassign x_h
    if CD_h is empty  then
        return (h, false)
    else
        return (h, true)
end  bt-unlabel
```

局所整合性 (Local Consistency)

1. ノード整合性 (Node Consistency)
2. アーク整合性 (Arc Consistency)
3. パス整合性 (Path Consistency)
4. m 次整合性 (m-Consistency)

整合アルゴリズム

- AC-1
- AC-2 (Waltz の フィルタリング, 記号的弛緩法)
- AC-3
- AC-4
- AC-5

ノード整合性 (Node Consistency)

- ノード i について, ($node\ consistent$) とは

$$\forall v\ [v \in D_i \longrightarrow C_i(v)]$$

- 制約グラフのすべてのノードがノード整合
  $\Longleftrightarrow$ 制約グラフがノード整合
- ノード整合アルゴリズム ($Node\ Consistency\ Algorithm$)
  ノード整合となるように, 各変数の領域から不要な値を除去しておく.

アーク整合性 (Arc Consistency)

- 有向アーク (i, j) に対し，$(arc\ consistent)$：

$$\forall v\ [v \in D_i \longrightarrow \exists w\ [w \in D_j \wedge C_{ij}(v, w)]]$$

- 制約グラフのすべてのノードがアーク整合
  $\Longleftrightarrow$ 制約グラフがアーク整合

最大アーク整合領域

- 制約グラフ G の P における P' は

$$P = D_1 \times D_2 \times \ldots \times D_n, P' = D'_1 \times D'_2 \times \ldots \times D'_n, \text{s.t.} P \supseteq P'$$

  - G は P' に関してアーク整合
  - $P \supseteq P'' \supset P'$ なる G に対してアーク整合となる P'' は存在しない．
- 【定理】最大アーク整合領域が存在し，かつ，ユニークである．

パス整合性 (Path Consistency)

- ノードの集まり $(i_0, i_1, \cdots, i_m)$ を通る長さ m のパス (経路) に対して, (m 次) $(path\ consistent)$：

$$\forall v \in D_{i_0} \forall u \in D_{i_m}\ [C_{i_0}(v) \wedge C_{i_m}(u) \wedge C_{i_0 i_m}(v, u)$$
$$\longrightarrow \exists y_1, \cdots, y_{m-1}\ [C_{i_1}(y_1) \wedge \cdots \wedge C_{i_{m-1}}(y_{m-1})$$
$$\wedge C_{i_0 i_1}(v, y_1) \wedge \cdots \wedge C_{i_{m-1} i_m}(y_{m-1}, u)]]]$$

- 0 次パス整合性 $\equiv$ ノード整合性
- 1 次パス整合性 $\equiv$ アーク整合性

m 次整合性 (m-Consistency)

- 制約グラフのすべての長さ m のパスに対して，k 次整合 $(k \leq m)$ していないパスが含まれていない時, $(m\text{-}Consistent)$ という.