人工知能特論： 局所整合性 (Local Consistency)　第 4 回

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，**ノード整合** ($node\ consistent$) 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，アーク整合 $(arc\ consistent)$：**

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

- **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**最大アーク整合領域** 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) パス整合 $(path\ consistent)$:**

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

- **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 次整合** (m-$Consistent$) という.

AC-5 ：アーク整合アルゴリズム

AC-5 は汎用 ($generic$) 整合アルゴリズム

- $ArcCons$ と $LocalArcCons$ という抽象化手続き使用
- AC-3 や AC-4 に特化可能
- 制約の特徴を活用するように特化可能
  - 関数型制約 (functional constraints)
  - 反関数型制約 (anti-functional constraints)
  - 単調型制約 (monotonic constraints)
  - 要素毎の関数型制約 (peicewise functional constraints)
  - 要素毎の反関数型制約 (peicewise anti-functional ...)
  - 要素毎の単調型制約 (peicewise monotonic constraints)

AC-5 の Queue 要素

AC-5 は $((i, j), w)$ という形式の $queue$ を管理.

- (i, j) はアーク, w は D_j から取り除かれた値. 後でアーク (i, j) を再検討する時に使用.
- $Enqueue(j, \Delta, Q)$ は, アーク (i, j) の $w \in \Delta$ なる queue Q に $((i, j), w)$ 形式の全要素を挿入.

k

j Δ は C_{ij} の無矛盾化のために D_j から除去した要素

i m

$ArcCons$ と $LocalArcCons$

function $ArcCons(i,j)$

Returns $\Delta = \{v \in D_i \mid \forall u \in D_j \ \neg C_{ij}(v,u)\}$

D_i から Δ の要素を除去し (i,j) を consistent にする

function $LocalArcCons(i,j,w)$

D_j から w が除去されていると仮定

Returns Δ s.t. $\Delta_2 \supseteq \Delta \supseteq \Delta_1$ where

$$\Delta_1 = \{v \in D_i \mid C_{ij} \ \wedge \ \forall u \in D_j \ \neg C_{i,j}(v,u)\}$$

$$\Delta_2 = \{v \in D_i \mid \forall u \in D_j \ \neg C_{i,j}(v,u)\}$$

AC-5 ：アーク整合アルゴリズム

procedure $AC\text{-}5(G)$

$InitQueue(Q)$

for each $(i,\ j) \in arc(G)$ **do**

Δ = $ArcCons(i,\ j)$

$Enqueue(i, \Delta, Q)$

$Remove(\Delta, D_i)$ **end**

while not $EmptyQueue(Q)$ **do**

$((i,\ j),\ w)$ = $Dequeue(Q)$

Δ = $LocalArcCons(i,\ j,\ w)$

$Enqueue(i, \Delta, Q)$

$Remove(\Delta, D_i)$ **end**

end $AC\text{-}5$

AC-5 での queue 演算回数

- 対 "$(edge,\ value)$" 対して，$Status$ を導入
  - $InitQueue$： $Status((k,\ i),\ v) =$ $present$ if $v \in D_i$
    $rejected$ otherwize
  - $EnQueue$： 各 queue 要素の $Status =$ $suspended$
  - $DeQueue$： 各 queue 要素の $Status =$ $rejected$
- AC-5 のループで保持される不変関係： $Status((k,\ i),\ v)$
  $=$ $present$ iff $v \in D_i$
  $=$ $suspended$ iff $v \notin D_i$ かつ $((k,\ i),\ v)$ が Q にある
  $=$ $rejected$ iff $v \notin D_i$ かつ $((k,\ i),\ v)$ が Q にない

$\Longrightarrow$ AC-5 での enqueue と dequeue は高々 $O(ed)$

AC-3 ：アーク整合アルゴリズム

- どの制約に対しても $ArcCons$ は $O(d^2)$
- AC-3 ： AC-5 と同じ.

  ただし, $LocalArcCons$ は $ArcCons$ を用いて実現

$\Longrightarrow$ AC-3 での enqueue と dequeue は$O(ed^3)$

AC-4 ：アーク整合アルゴリズム

- $ArcCons$:$O(d^2)$, $LocalArcCons$:$O(d)$ $\Rightarrow$ **AC-5:** $O(ed^2)$

	i	j		
支 **2**	u	u	$\{u\}$	**支持**
持 **2**	v	v	$\{u, v\}$	**リスト**
数 **1**	w	w	$\{v, w\}$	
edge 辺りの時間	$O(d^2)$	$O(d^2)$		
edge 辺りの空間	$O(d)$	$O(d^2)$		

- $LocalArcCons(i, j, w)$ **は edge** (i, j) **に対して** w **の「支持リスト」で繰り返し，「支持数」を 1 減らす. 「支持数」が 0 になる値の集合として** Δ **を計算.**

$\Longrightarrow$ $ArcCons$:$O(d^2)$, $LocalArcCons$:$O(d)$ $\Rightarrow$ **AC-4:** $O(ed^2)$

関数型制約 (functional constraints)

制約 C が領域 D に関して関数的とは、 iff $\forall v \in D$ に対して

$\{w \in D | C(v,w)\}$ なる w が高々 1 つ存在

function $ArcCons(i,j)$

$\Delta = \{\}$

for each $v \in D_i$ `bf do`

if $f_{ij}(v) \notin D_j$ **then** $\Delta = \Delta \cup \{v\}$

return Δ

end $ArcCons$

$ArcCons$ は $O(d)$

$LocalArcCons$ は $O(1)$

AC-5 は $O(ed)$

function $LocalArcCons(i,j,w)$

if $f_{ij}(v) \in D_j$ **then return** $\{f_{ij}(w)\}$

else return $\{\}$

end $LocalArcCons$

アルゴリズム $REVISE$ — $ArcCons$ とほぼ同じ

procedure $REVISE((i,\ j))$
begin
    $DELETE$ = **false**
    for each $v \in D_i$ **do**
        if $C_{ij}(v, u)$ なる $u \in D_j$ が存在しない **then**
            begin
                $Remove(x, D_i)$
                $DELETE$ = **true**
            end
    return $DELETE$
end $REVISE$

AC-1 ：アーク整合アルゴリズム

procedure $AC\text{-}1(G)$

begin

Q = $\{(i,\ j)\ |\ (i,\ j) \in arc(G)\ ,\ i \neq j\}$

repeat

begin

$CHANGE$ = **false**

for each $(i,\ j) \in Q$ **do**

$CHANGE$ = $REVISE((i,\ j)) \vee CHANGE$

end

until $\neg\ CHANGE$

end $AC\text{-}1$

AC-1 の時間計算量

- どの制約に対しても $REVISE$ は C_{ij} を$O(d^2)$ 回計算
- repeat 中で $|Q| = 2e$
- repeat での最悪のケースは，1 回に 1 個しか変数が削除されない場合.

  つまり, repeat が d 回繰り返される.

$\Longrightarrow$ AC-1 での最悪計算量は $O(ed^3)$

AC-2 ：アーク整合アルゴリズム

procedure $AC\text{-}2(G)$

begin

  for i = 1 **to** n **do**

    Q = $\{(i,j) \mid (i,j) \in arc(G),\ i < j\}$

    Q' = $\{(j,i) \mid (j,i) \in arc(G),\ j < i\}$

    while not $EmptyStack(Q)$ **do**

      while not $EmptyStack(Q)$ **do**

        (k,m) = $pop(Q)$

        if $REVISE((k,m))$ **then**

          $\mathtt{push}(Q', \{(p,k) \mid (p,k) \in arc(G), p \leq i, p \neq m\})$

      end

      Q = Q' ; Q' = $empty$

    end

  end

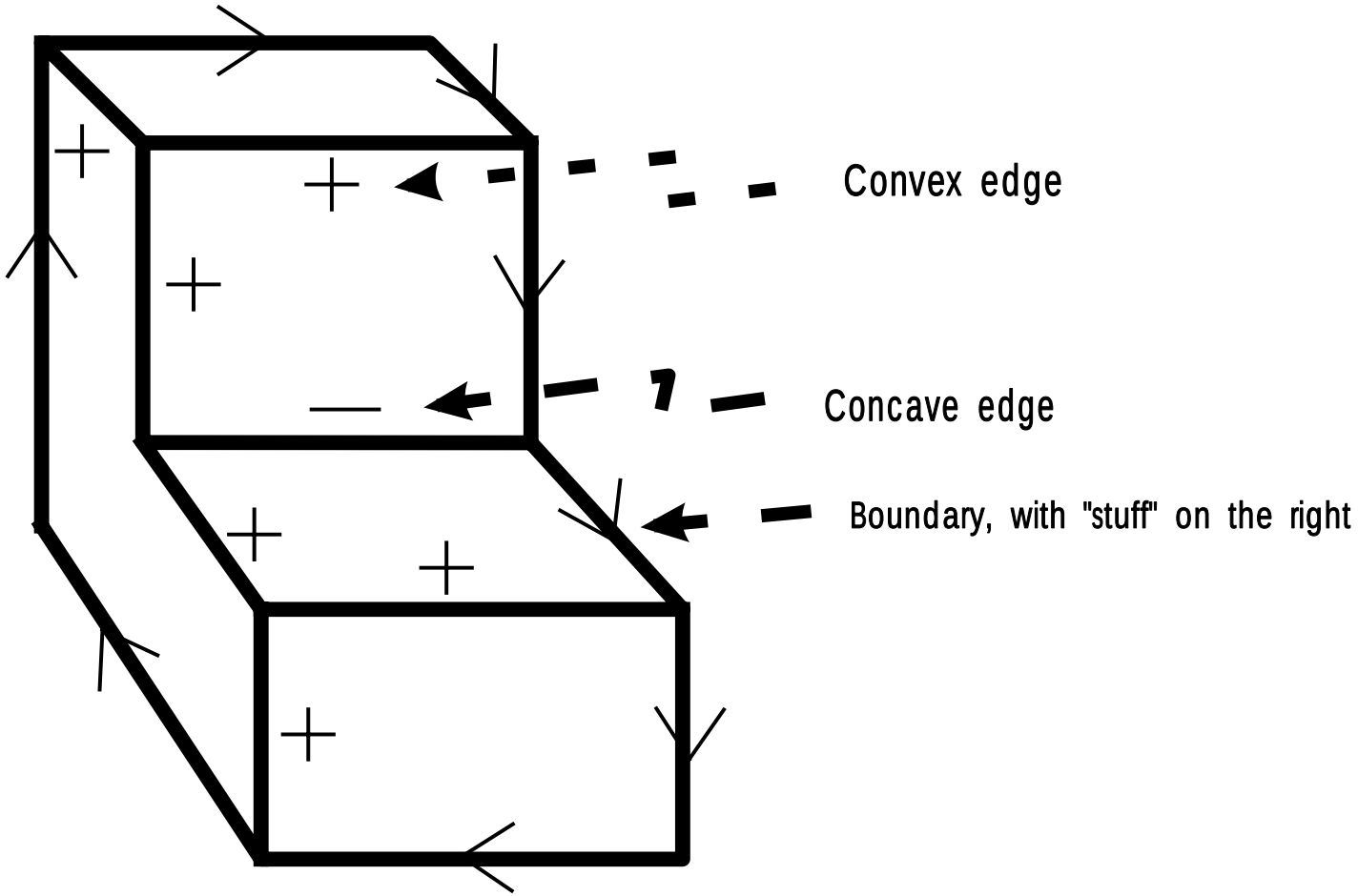
end $AC\text{-}2$

AC-2 の時間計算量

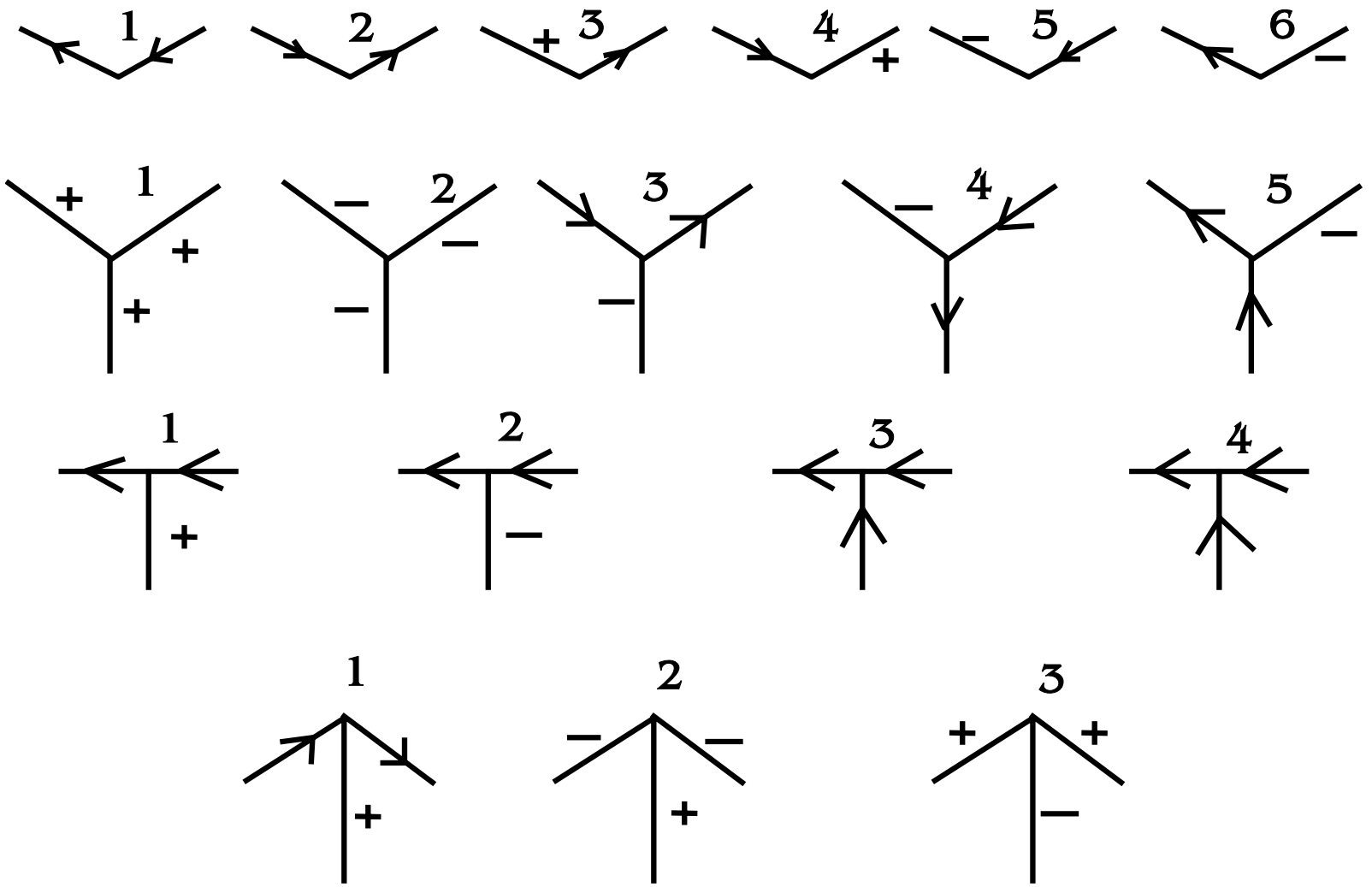
- AC-2 での最悪計算量は $O(ed^3)$
- Waltz の (フィルタリング) アルゴリズム
- 制約伝播法 (Constraint propagation) に基づいた **記号的弛緩法** (*symbolic relaxation method*) とも呼ばれる

記号的弛緩法 あるいは Waltz フィルタリング

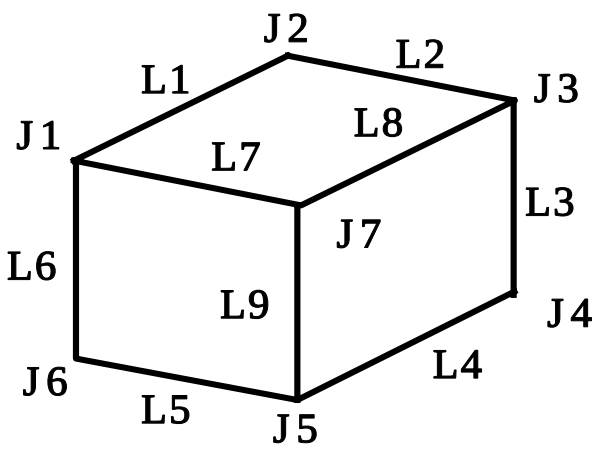
線画のラベル付け, 線ラベルの意味

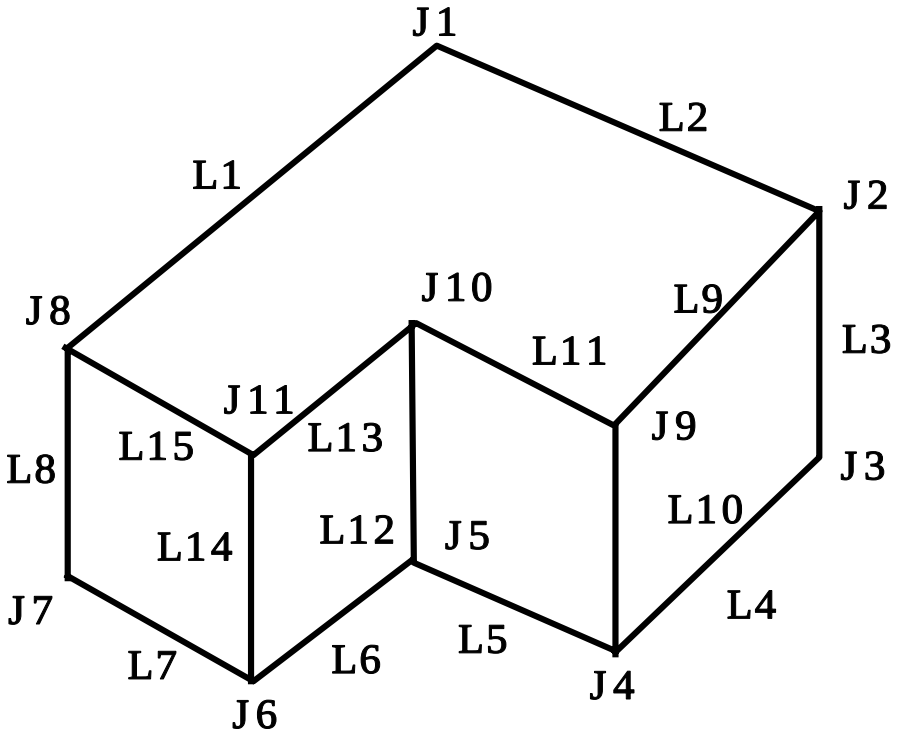


単純な積木の世界のジャンクション・カタログ [Winston]



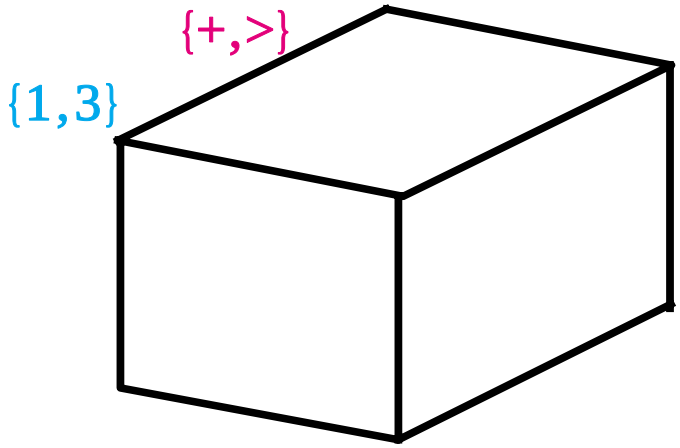
2 つの例で解析を行う


J2
L1
L2
J3
J1
L8
L7
L3
J7
L6
L9
J4
L4
J6
L5
J5

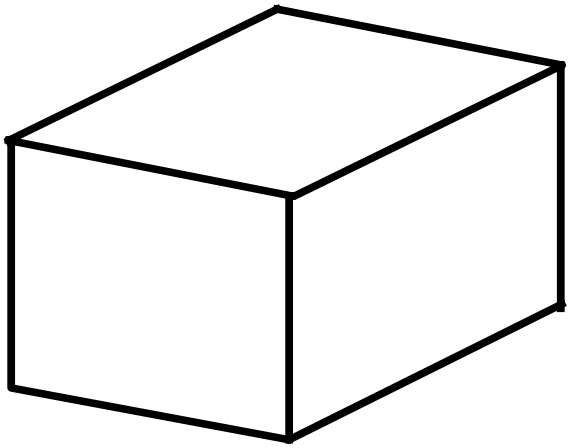

J1
L2
L1
J2
J10
L9
J8
L3
L11
J11
J9
L15
L13
L8
J3
L10
L14
L12
J5
J7
L4
L5
L7
L6
J4
J6

立方体の解析：可能性と無矛盾な解釈

可能なラベルつけ

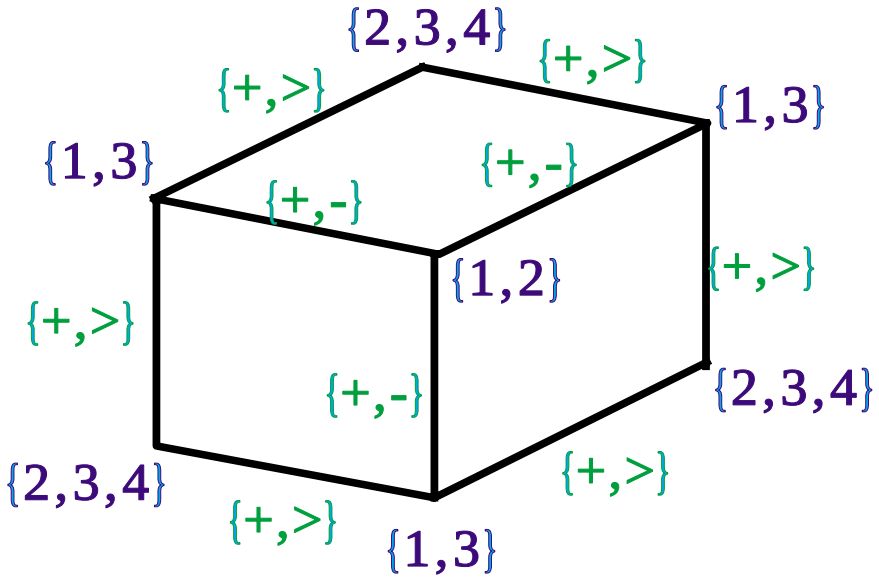


無矛盾な解釈

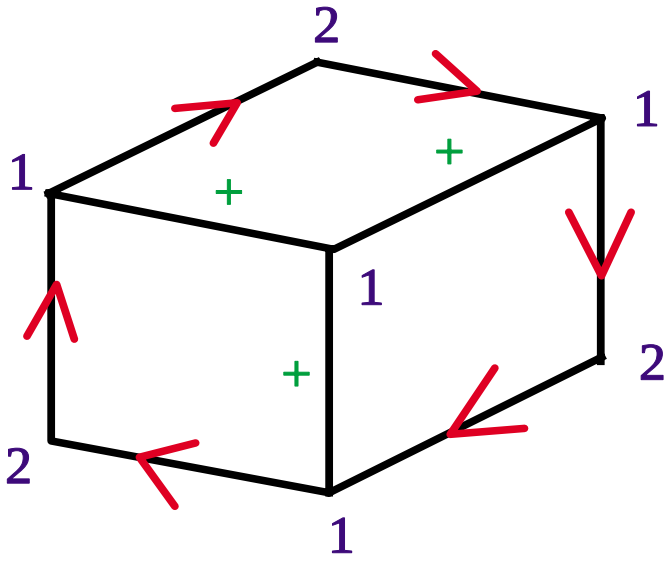


立方体の解析：可能性と無矛盾な解釈

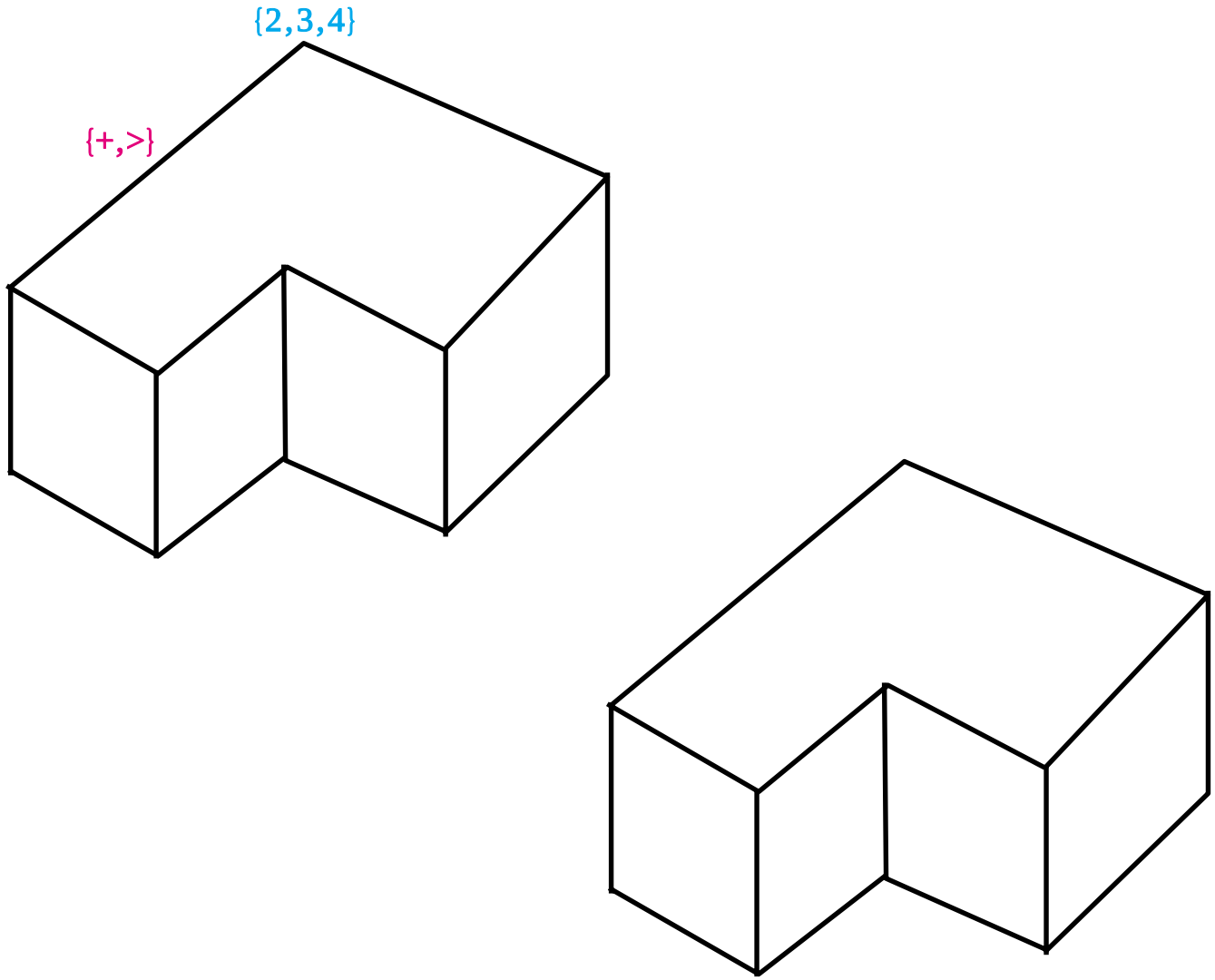
可能なラベルつけ



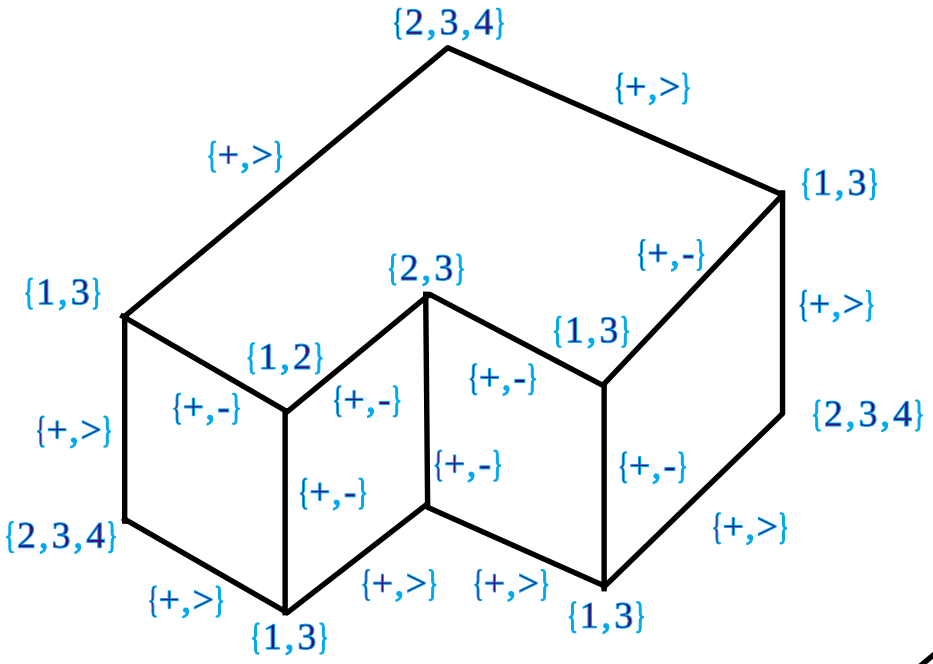
無矛盾な解釈

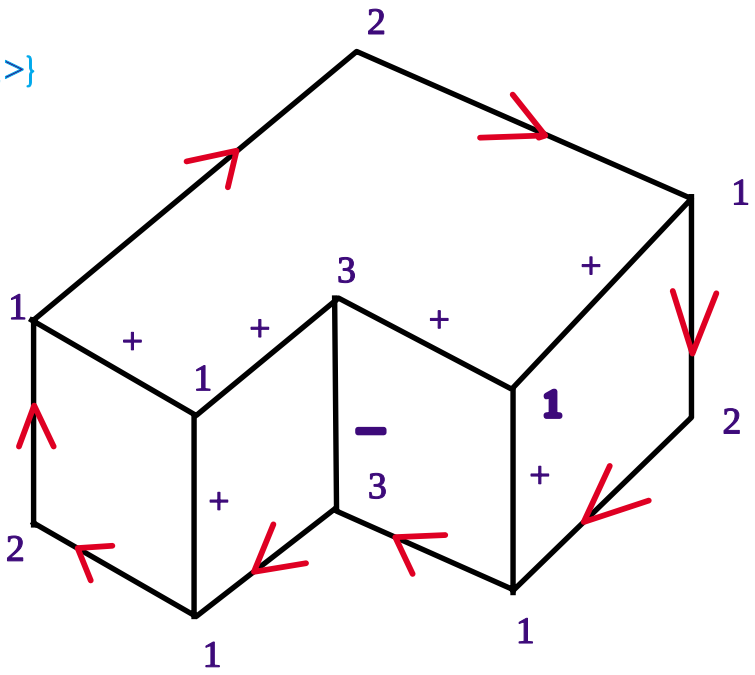


へこみのある物体の解析：可能性と無矛盾な解釈



へこみのある物体の解析：可能性と無矛盾な解釈





AC-3 ：アーク整合アルゴリズム

procedure $AC\text{-}3(G)$

begin

$Q = \{(i,j) \mid (i,j) \in arc(G),\ i \neq j\}$

while not $EmptyQueue(Q)$ **do**

begin

$(k,m) = pop(Q)$

if $REVISE((k,m))$ **then**

$Q = Q \cup \{(i,k) \mid (i,k) \in arc(G), i \neq k, i \neq m\}$

end

end

end $AC\text{-}3$

AC-3 の時間計算量

- AC-2：深さ優先探索，AC-3：幅優先探索
- $REVISE$ が成立する時 — ノード k にアークの個数 d_k とすると、高々 $d_k - 1$ 個アークが Q に追加.

$$\sum_{k=1}^{n} d(d_k - 1) = d(2e - n)$$

- repeat 中で少なくとも 1 個の値が除去

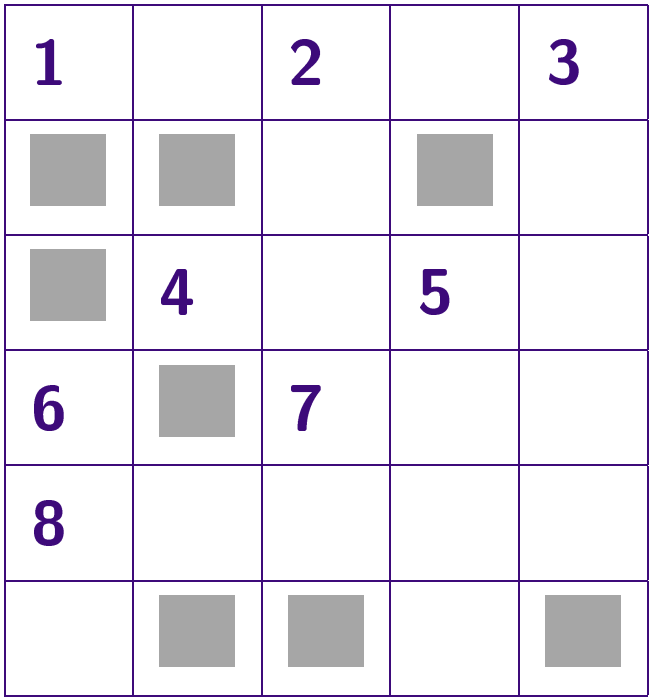
  繰り返しの個数は Q の長さ，つまり，$2e + d(2e - n)$
- 各繰り返しでは，d^2 回の C_{ij} の評価が必要

$\Longrightarrow$ AC-3 での最悪計算量は $O(ed^3)$

下限： $REVISE$ がすべて偽 $\Longrightarrow \Omega(ed^2)$

課題 3 クロスワードパズル

AC-3 以上のアーク整合アルゴリズムを使用して解きなさい.



Word List

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