Constraint Programming e sue applicazioni alla Predizione di Struttura

Agostino Dovier

Università di Udine
Dipartimento di Matematica e Informatica

Udine, 8 Maggio 2008

General Information

- Due to rector elections, lectures number 5 and 6 have been scheduled Friday 16 (9:00 and 11:00)
- All course slides are available at the course web page `http://www.dimi.uniud.it/dovier/DID/vincoliNMR.html`
- Other material (e.g. a 250 pages free book and several links) can be found in the web page of my course of Modern Languages (Linguaggi di Nuova Concezione) `http://www.dimi.uniud.it/dovier/LNC/prog.html`

Sommario

CONSTRAINT SOLVER

Un risolutore di vincoli è una procedura che trasforma un CSP $\mathcal{P}$ in uno equisoddisfacibile (o equivalente). Si dice:

COMPLETO se, dato $\mathcal{P}$, lo trasforma in un CSP o in una disgiunzione finita di CSP ad esso equisoddisfacibile, e tale che da ciascuno dei disgiunti sia immediato trarre ogni sua soluzione; se $\mathcal{P}$ è inconsistente, viene restituito `fail`.

INCOMPLETO se non è completo. Intuitivamente, dato $\mathcal{P}$, lo trasforma in un CSP più semplice ma non ancora abbastanza semplice.

PROOF RULES

Un constraint solver è basato sull'applicazione ripetuta di *regole di dimostrazione*

$$\frac{\varphi}{\psi}$$

ove φ e ψ sono dei CSP. In ψ possono occorrere anche variabili non presenti in φ. Una regola mantiene l'equivalenza (è *equivalence preserving*) se $Sol(\varphi) = Sol(\psi)|_{FV(\varphi)}$, ovvero φ e ψ sono equisoddisfacibili.

Si vuole restringere i domini sfruttando le informazioni presenti nei vincoli. Le regole hanno la forma:

$$\frac{\varphi = \langle \mathcal{C}; \mathcal{D}_{\in} \rangle}{\psi = \langle \mathcal{C}'; \mathcal{D}'_{\in} \rangle}$$

- $\mathcal{D}_{\in} = X_1 \in \mathcal{D}_1, \ldots, X_k \in \mathcal{D}_k$,
- $\mathcal{D}'_{\in} = X_1 \in \mathcal{D}'_1, \ldots, X_k \in \mathcal{D}'_k$,
- per ogni $i = 1, \ldots, k$ vale che $\mathcal{D}'_i \subseteq \mathcal{D}_i$, e
- $\mathcal{C}'$ è la restrizione di $\mathcal{C}$ ai nuovi domini delle variabili.

Alcuni vincoli in $\mathcal{C}'$ possono diventare ridondanti e dunque rimossi. Se tutti i vincoli si possono rimuovere, $\mathcal{C}' = \texttt{true}$ ($\mathcal{D}'_{\in}$ rappresentate tutte le soluzioni).
Se invece un $\mathcal{D}'_i = \emptyset$ allora $\psi = \texttt{fail}$.

$$\frac{\langle Y < X; X \; in \; 0..10, Y \; in \; 5..15 \rangle}{\langle Y < X; X \; in \; 6..10, Y \; in \; 5..9 \rangle}$$

Abbiamo che:

$$\begin{aligned} \mathcal{C} = \mathcal{C}' = \; \{ \quad & (5,6),(5,7),(5,8),(5,9),(5,10), \\ & (6,7),(6,8),(6,9),(6,10), \\ & (7,8),(7,9),(7,10), \\ & (8,9),(8,10),(9,10)\} \end{aligned}$$

Constraint Solving
Transformation Rules

I vincoli sono trasformati/semplificati. È possibile che la semplificazione introduca nuove variabili a cui va assegnato un dominio. Può essere introdotto il vincolo `fail`.

$$\frac{\varphi = \langle \mathcal{C}; \mathcal{D}_\in \rangle}{\psi = \langle \mathcal{C}'; \mathcal{D}'_\in \rangle}$$

- $\mathcal{D}_\in = X_1 \in \mathcal{D}_1, \ldots, X_k \in \mathcal{D}_k$,
- $\mathcal{D}'_\in = X_1 \in \mathcal{D}_1, \ldots, X_k \in \mathcal{D}_k$, più eventualmente nuovi domini non vuoti per nuove variabili (i domini delle variabili già presenti non si riducono)

$$\frac{\langle e_1 \neq e_2; \mathcal{D}_\in \rangle}{\langle X = e_1, X \neq e_2; \mathcal{D}_\in, X \; in \; \mathbb{Z} \rangle}$$

Viene introdotta una nuova variabile, il cui dominio iniziale è il più grande possibile (assumiamo di lavorare con numeri interi). L'introduzione di una nuova variabile rende necessario ricorrere al concetto di equisoddisfacibilità tra CSP.

In alcuni casi risulta necessario o utile introdurre nuovi vincoli (tipicamente implicati dai vincoli già presenti). Le regole hanno la forma:

$$\frac{\varphi = \langle \mathcal{C}; \mathcal{D}_{\in} \rangle}{\psi = \langle \mathcal{C}, \mathcal{C}; \mathcal{D}_{\in} \rangle}$$

dove C è un nuovo constraint.

Consideriamo questa regola con domini sui reali.

$$\frac{\langle X^2 - 2XY + Y^2 < 5; X > 0, Y > 0\rangle}{\langle X^2 - 2XY + Y^2 < 5, X - Y < 3; X > 0, Y > 0\rangle}$$

Il nuovo vincolo non aggiunge informazione essendo implicato dal primo CSP. Tuttavia introdurre vincoli semplici, se pur ridondanti, può rendere più efficiente il processo di inferenza di un constraint solver.

- L'applicazione ripetuta delle regole dei tre tipi visti viene detta fase di propagazione di vincoli.
- Una *derivazione* è una sequenza di applicazioni delle regole di dimostrazione, previa opportuna rinomina delle variabili presenti nelle regole.
- Una derivazione finita è:
  - di *fallimento* se l'ultimo CSP è `fail`,
  - *stabile* se non è di fallimento e si raggiunge un CSP per cui nessuna regola è applicabile,
  - di *successo* se è stabile e l'ultimo CSP è in forma risolta (ovvero è consistente e da esso è immediato individuare una soluzione).

La fase di constraint propagation deve essere semplice computazionalmente (P).

NODE CONSISTENCY

Un CSP $\mathcal{P}$ è *node consistent* se per ogni variabile X in $\mathcal{P}$, ogni vincolo unario relativo a X coincide con il dominio di X.

$$\langle X_1 \geq 0, X_2 \geq 0; X_1 \ in \ \mathbb{N}, X_2 \ in \ \mathbb{N} \rangle$$

è node consistent

$$\langle X_1 \geq 5, X_2 \geq 1; X_1 \ in \ \mathbb{N}, X_2 \ in \ \mathbb{N} \rangle$$

non lo è (è comunque consistente)

$$\langle X_1 \neq X_2, X_1 = 0, X_2 = 0; X_1 \ in \ 0..0, X_2 \ in \ 0..0 \rangle$$

è node consistent, ma non è consistente

Constraint Propagation
Node Consistency

La node consistency si ottiene applicando ripetutamente la seguente regola di *domain reduction*:

$$\frac{\langle C; X \text{ in } D_X \rangle}{\langle C; X \text{ in } D_X \cap C \rangle}$$

dove C è un vincolo unario sulla variabile X.

ARC CONSISTENCY

Un constraint binario C sulle variabili X ed Y aventi rispettivi domini D_X e D_Y è *arc consistent* se:

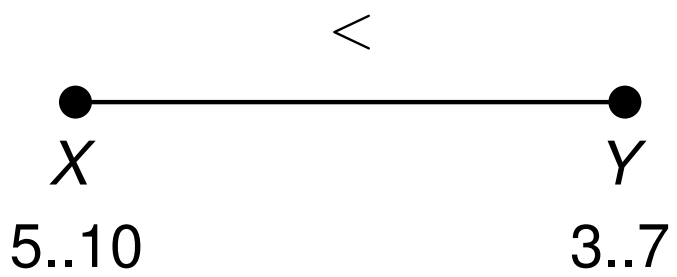
1. $(\forall a \in D_X)(\exists b \in D_Y)\,((a,b) \in C)$, e
2. $(\forall b \in D_Y)(\exists a \in D_X)\,((a,b) \in C)$.

Un CSP $\langle \mathcal{C}; \mathcal{D}_{\in} \rangle$ è arc consistent se lo sono tutti i constraint binari presenti in $\mathcal{C}$.

Le nozioni di arco e nodo vanno fatte risalire al grafo che possiamo costruire mettendo come nodi le variabili di un CSP (etichettati dai loro domini) e come archi i vincoli binari tra queste.

$$\langle X < Y; X \; in \; 5..10, Y \; in \; 3..7 \rangle$$

si può rappresentare con il grafo:



Non è arc consistent ma è consistente.

$$\langle X \neq Y, Y \neq Z, X \neq Z; X \; in \; 0..1, Y \; in \; 0..1, Z \; in \; 0..1 \rangle$$

è arc consistent, ma non consistente.

L'arc consistency si ottiene applicando ripetutamente due regole di *domain reduction*):

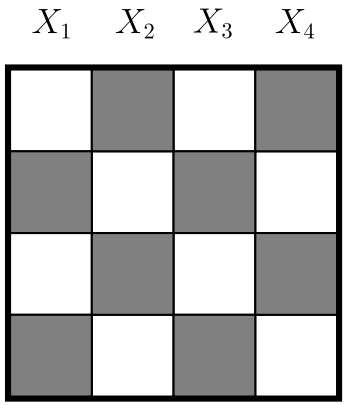
$$(AC1) \qquad \frac{\langle C; X \text{ in } D_X, Y \text{ in } D_Y \rangle}{\langle C; X \text{ in } D'_X, Y \text{ in } D_Y \rangle}$$

$$(AC2) \qquad \frac{\langle C; X \text{ in } D_X, Y \text{ in } D_Y \rangle}{\langle C; X \text{ in } D_X, Y \text{ in } D'_Y \rangle}$$

dove C è un vincolo binario sulle variabili X ed Y, mentre:

- $D'_X = \{a \in D_X \ : \ (\exists b \in D_Y)((a,b) \in C)\}$
- $D'_Y = \{b \in D_Y \ : \ (\exists a \in D_X)((a,b) \in C)\}$

Constraint Propagation
Arc Consistency, $X_1 = 1$

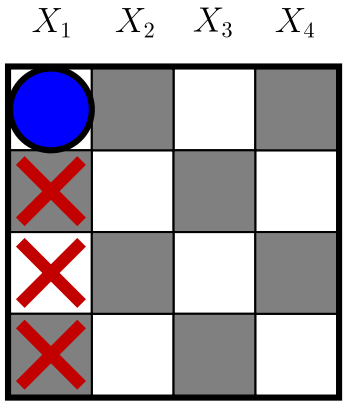


$$X_1, \ldots, X_4$$

$$D(X_i) = \{1, \ldots, 4\} \text{ per } i = 1..4$$

$$X_i \neq X_j, |X_i - X_j| \neq (j - i) \text{ per } i < j$$

Constraint Propagation
Arc Consistency, $X_1 = 1$

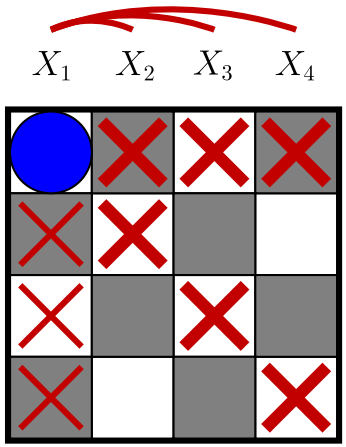


$$X_1, \ldots, X_4$$

$$D(X_1) = \{1\}, D(X_i) = \{1, \ldots, 4\} \text{ per } i = 2..4$$

$$X_i \neq X_j, |X_i - X_j| \neq (j - i) \text{ per } i < j$$

Constraint Propagation
Arc Consistency, $X_1 = 1$

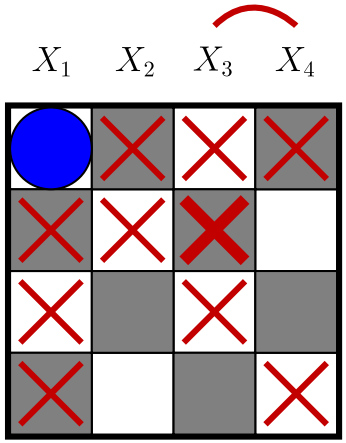


$$X_1, \ldots, X_4$$

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

$$X_i \neq X_j, |X_i - X_j| \neq (j - i) \text{ per } i < j$$

Constraint Propagation
Arc Consistency, $X_1 = 1$

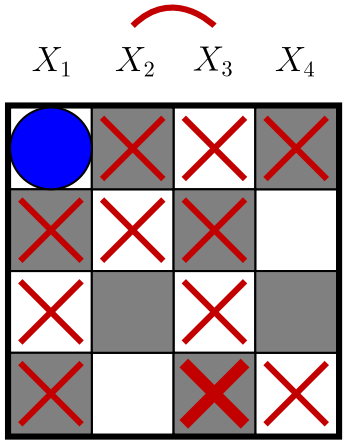


$$X_1, \ldots, X_4$$

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

$$X_i \neq X_j, |X_i - X_j| \neq (j - i) \text{ per } i < j$$

Constraint Propagation
Arc Consistency, $X_1 = 1$

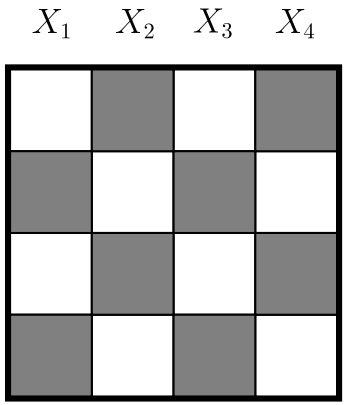


$$X_1, \ldots, X_4$$

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

$$X_i \neq X_j, |X_i - X_j| \neq (j - i) \text{ per } i < j$$

Constraint Propagation
Arc Consistency: $X_1 = 2$

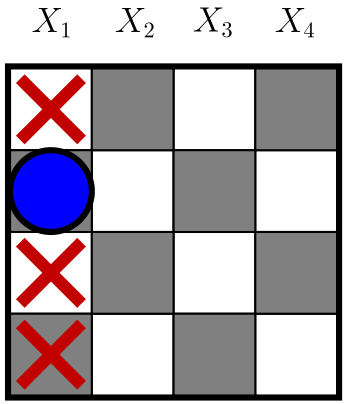


$$X_1, \ldots, X_4$$

$$D(X_i) = \{1, \ldots, 4\} \text{ per } i = 1..4$$

$$X_i \neq X_j, |X_i - X_j| \neq (j - i) \text{ per } i < j$$

Constraint Propagation
Arc Consistency: $X_1 = 2$

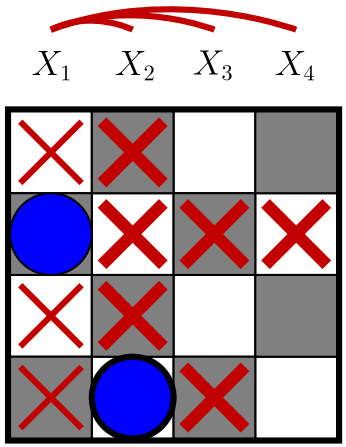


$$X_1, \ldots, X_4$$

$$D(X_1) = \{2\}, D(X_i) = \{1, \ldots, 4\} \text{ per } i = 2..4$$

$$X_i \neq X_j, |X_i - X_j| \neq (j - i) \text{ per } i < j$$

Constraint Propagation
Arc Consistency: $X_1 = 2$

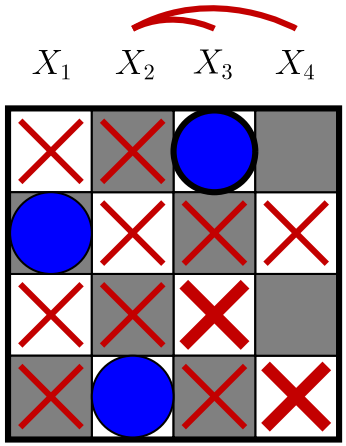


$$X_1, \ldots, X_4$$

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

$$X_i \neq X_j, |X_i - X_j| \neq (j - i) \text{ per } i < j$$

Constraint Propagation
Arc Consistency: $X_1 = 2$

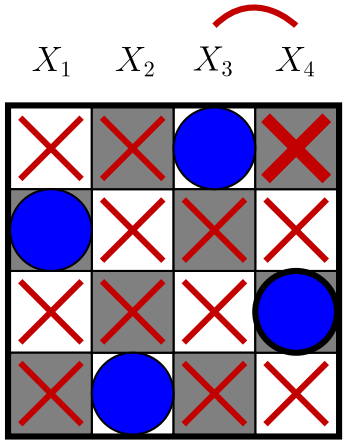


$$X_1, \ldots, X_4$$

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

$$X_i \neq X_j, |X_i - X_j| \neq (j - i) \text{ per } i < j$$

Constraint Propagation
Arc Consistency: $X_1 = 2$



$$X_1, \ldots, X_4$$

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

$$X_i \neq X_j, |X_i - X_j| \neq (j - i) \text{ per } i < j$$

BOUNDS CONSISTENCY

Un constraint binario C sulle variabili X ed Y aventi rispettivi domini D_X e D_Y è *bounds consistent* se:

1. $(\exists b \in \min(D_Y)..\max(D_Y))\,((\min(D_X), b) \in C)$, e
   $(\exists b \in \min(D_Y)..\max(D_Y))\,((\max(D_X), b) \in C)$,
2. $(\exists a \in \min(D_X)..\max(D_X))\,((a, \min(D_Y)) \in C)$, e
   $(\exists a \in \min(D_X)..\max(D_X))\,((a, \max(D_Y)) \in C)$,

Un CSP $\langle \mathcal{C}; \mathcal{D}_{\in} \rangle$ è bounds consistent se lo sono tutti i constraint binari presenti in $\mathcal{C}$.

Il seguente CSP è consistent, bounds consistent, ma non arc consistent:

$$\langle 2X = Y; X \; in \; \{0,1,2\}, Y \; in \; \{0,4\} \rangle$$

Applicando (AC1) si otterrebbe infatti:

$$\langle 2X = Y; X \; in \; \{0,2\}, Y \; in \; \{0,4\} \rangle$$

Le due regole per la bounds consistency sono:

$$(BC1) \qquad \frac{\langle C; X \text{ in } D_X, Y \text{ in } D_Y\rangle}{\langle C; X \text{ in } D'_X, Y \text{ in } D_Y\rangle}$$

$$(BC2) \qquad \frac{\langle C; X \text{ in } D_X, Y \text{ in } D_Y\rangle}{\langle C; X \text{ in } D_X, Y \text{ in } D'_Y\rangle}$$

- $D'_X = D_X \cap (\min(\{a \in D_X \ : \ (\exists b \in \min(D_Y)..\max(D_Y))((a,b) \in C)\})..\max(\{a \in D_X \ : \ (\exists b \in \min(D_Y)..\max(D_Y))((a,b) \in C)\}))$
- $D'_Y = D_Y \cap (\min(\{b \in D_Y \ : \ (\exists a \in \min(D_X)..\max(D_X))((a,b) \in C)\})..\max(\{b \in D_Y \ : \ (\exists a \in \min(D_X)..\max(D_X))((a,b) \in C)\}))$

Sembrano complesse. In realtà si possono implementare in tempo pressocché costante.

Si tratta di una generalizzazione dell'arc consistency a vincoli non binari.

HAC

Un vincolo n-ario C sulle variabili $X_1, \ldots, X_n$ è *hyper arc consistent* se per ogni $i = 1, \ldots, n$ vale che:

- $(\forall a_i \in D_i)(\exists a_1 \in D_1) \cdots (\exists a_{i-1} \in D_{i-1})(\exists a_{i+1} \in D_{i+1}) \cdots (\exists a_n \in D_n)(\langle a_1, \ldots a_n \rangle \in C)$

Un CSP è *hyper arc consistent* se lo sono tutti i suoi vincoli.

$$\langle 2X + 3Y < Z; X \; in \; 1..10, Y \; in \; 1..10, Z \; in \; 1..10 \rangle$$

Qualunque coppia di valori per X ed Y garantisce almeno la somma 5. Dunque, essendoci 1 in D_Z il CSP non è hyper arc consistent. Anche il 3 in D_Y è un valore da togliere. Infatti, con tale valore l'espressione a sinistra arriva a 9, a cui va sommato almeno 2 come contributo di X. Ció supera il valore massimo a sinistra (9).
Il CSP equivalente e hyper arc consistent è:

$$\langle X + Y < Z; X \; in \; 1..3, Y \; in \; 1..2, Z \; in \; 6..10 \rangle$$

Per ottenere questa proprietà vanno applicate le regole (*HACi*) di seguito descritte, ove $i = 1, \ldots, n$:

$$\frac{\langle C; X_1 \text{ in } D_1, \ldots, X_{i-1} \text{ in } D_{i-1}, X_i \text{ in } D_i, X_{i+1} \text{ in } D_{i+1}, \ldots, X_n \text{ in } D_n \rangle}{\langle C; X_1 \text{ in } D_1, \ldots, X_{i-1} \text{ in } D_{i-1}, X_i \text{ in } D'_i, X_{i+1} \text{ in } D_{i+1}, \ldots, X_n \text{ in } D_n \rangle}$$

ove C è un vincolo su $a_1, \ldots, a_n$ e

$$\begin{aligned} D'_i \quad = \quad & \{a_i \in D_i (\exists a_1 \in D_1) \cdots (\exists a_{i-1} \in D_{i-1}) \\ & (\exists a_{i+1} \in D_{i+1}) \cdots (\exists a_n \in D_n)(\langle a_1, \ldots a_n \rangle \in C)\} \end{aligned}$$

Per ogni vincolo n-ario vi saranno dunque n di queste regole proiettive.

Constraint Propagation
Altre regole

- Directional arc/bounds consistency
- Hyper bounds consistency
- Directional HAC, BAC
- Path Consistency
- k-consistency

- In Constraint Programming vincoli che riguardano gruppi di variabili (in pratica, tutti i vincoli non unari nè binari) sono detti vincoli globali.
- Solitamente vincoli globali possono essere implementati come congiunzioni di vincoli binari, ma in tal caso la propagazione che si ottiene è spesso molto povera.
- Vincoli globali di uso comune sono studiati indipendentemente.

- Dato un vincolo C su n-variabili $X_1, \ldots, X_n$ siamo interessati a mantenere/ottenere la proprietà di *hyper arc consistency*.
- Per molti vincoli globali la sola verifica di tale proprietà è NP-hard.
- In tal caso la propagazione farà una semplificazione (in tempo polinomiale) corretta dei vincoli (togliendo punti inutili dai domini) ma non è detto raggiunga l'HAC.
- Un caso elegante è il vincolo di `alldifferent`

alldifferent

Siano $X_1, \ldots, X_k$ variabili con domini rispettivi $\mathcal{D}_1, \ldots, \mathcal{D}_k$.
Il vincolo k-ario alldifferent$(X_1, \ldots, X_k)$ è definito come:

$$\text{alldifferent}(X_1, \ldots, X_k) = (\mathcal{D}_1 \times \cdots \times \mathcal{D}_k) \setminus \{(a_1, \ldots, a_k) \in \mathcal{D}_1 \times \cdots \times \mathcal{D}_k \ : \ \exists i \exists j \, 1 \leq i < j \leq k \, (a_i = a_j)\}$$

È *hyper arc consistent* se per ogni $i \in \{1, \ldots, k\}$ e per ogni $a_i \in \mathcal{D}_i$ vale che esistono $a_1 \in \mathcal{D}_1, \ldots, a_{i-1} \in \mathcal{D}_{i-1}, a_{i+1} \in \mathcal{D}_{i+1}, \ldots, a_k \in \mathcal{D}_k$ che soddisfano alldifferent$(X_1, \ldots, X_k)$.
Un CSP è *diff-arc consistent* se ogni vincolo di differenza in esso è *hyper arc consistent*.

Constraint Propagation
alldifferent

- $\langle \text{alldifferent}(X_1, \ldots, X_k); \mathcal{D}_\in \rangle$ e
- $\langle X_1 \neq X_2, X_1 \neq X_3, \ldots, X_1 \neq X_k, X_2 \neq X_3, \ldots, X_{k-1} \neq X_k; \mathcal{D}_\in \rangle$ sono equivalenti. *AC* del secondo non implica però *HAC* del primo (si pensi a $\langle \text{alldifferent}(X_1, X_2, X_3); D_1 = D_2 = D_3 = \{0, 1\} \rangle$).
- Usando tecniche sofisticate di teoria dei grafi bipartiti si implementa in modo efficiente la HAC (e anche la HBC) per il vincolo alldifferent.
- In generale, CP batte in efficienza altri approcci se ha buone implementazioni per la propagazione (anche approssimata) dei vincoli globali.

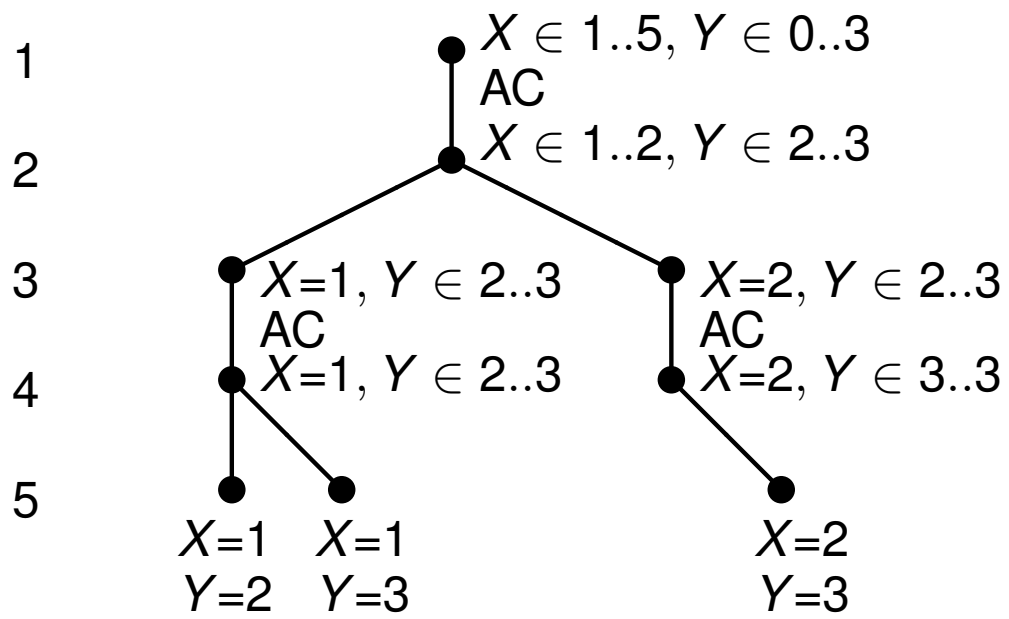
CONSTRAINT SOLVING

PROP-LABELING-TREE

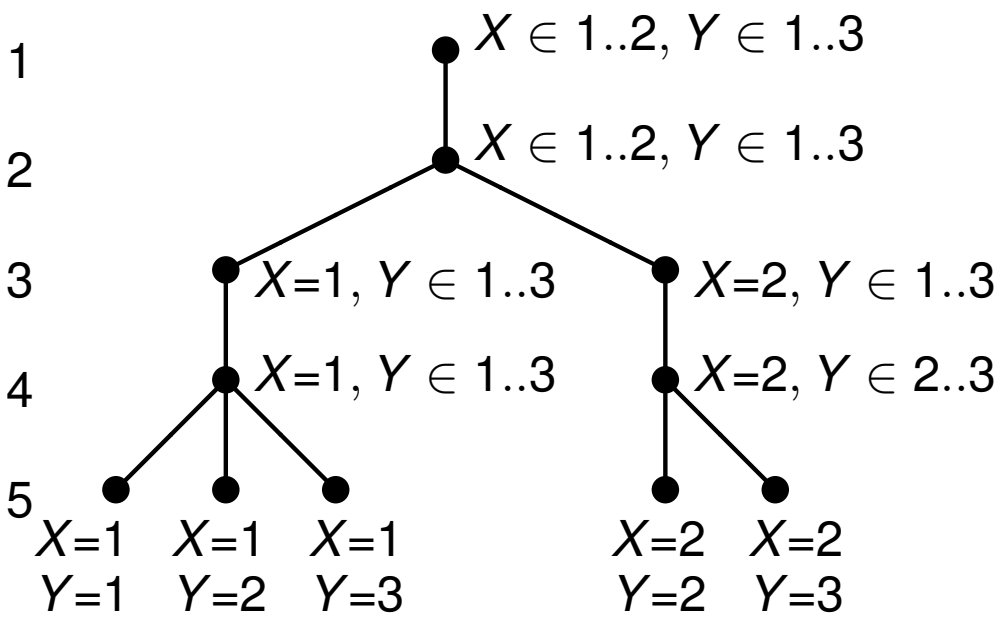
- La ricerca delle soluzioni avviene alternando fasi di scelta non-deterministica a fasi di propagazione deterministica.
- Si sceglie (in base a qualche regola) una variabile e si sceglie un valore del dominio (enumeration ... si può anche fare altro come domain splitting, indispensabile nel continuo)
- Si applica propagazione e si continua.
- Se un dominio diventa vuoto, si ferma quel ramo.
- Se non ci sono piu' variabili, si fornisce la soluzione.
- Grandezza e forma dell'albero dipendono dai criteri di scelta.

Constraint Solving

prop-labeling-tree



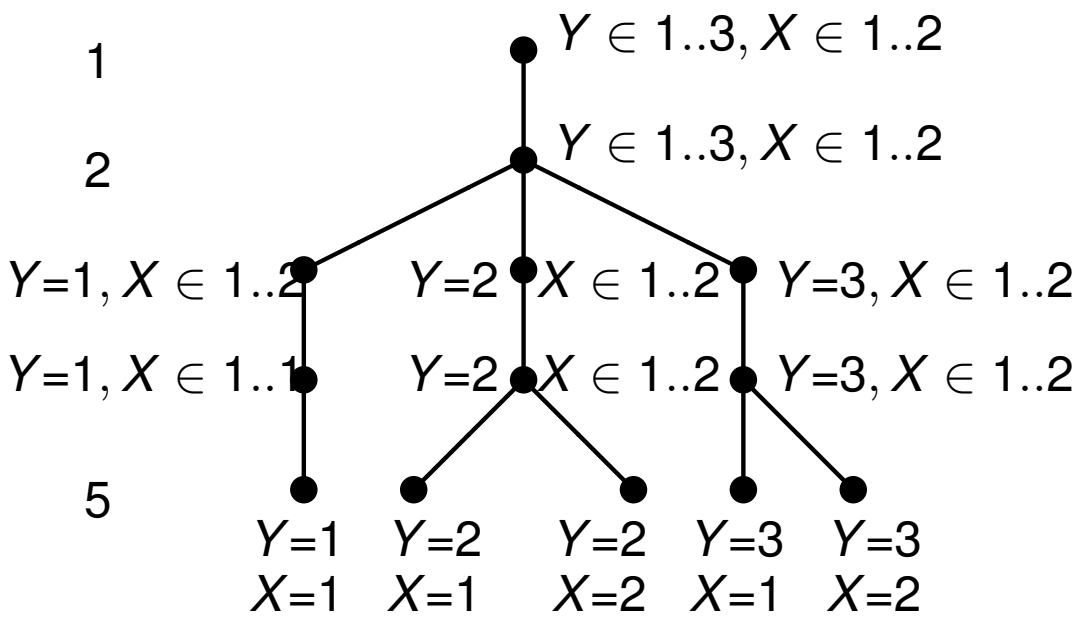
prop-labeling-tree per
$\mathcal{P} = \langle X < Y; X \in \{1,2,3,4,5\}, Y \in \{0,1,2,3\} \rangle$.



prop-labeling-tree per
$\mathcal{P} = \langle X \leq Y; X \in \{1,2\}, Y \in \{1,2,3\} \rangle$.

Constraint Solving

prop-labeling-tree



prop-labeling-tree per
$\mathcal{P} = \langle X \leq Y; X \in \{1,2\}, Y \in \{1,2,3\} \rangle$.

1. (domain) labeling:

$$\frac{X \in \{a_1, \ldots, a_k\}}{X \in \{a_1\} | \cdots | X \in \{a_k\}}$$

2. (domain) enumeration:

$$\frac{X \in \mathcal{D}}{X \in \{a\} | X \in \mathcal{D} \setminus \{a\}}$$

   ove $a \in \mathcal{D}$

3. (domain) bisection:

$$\frac{X \in \mathcal{D}}{X \in \min(\mathcal{D})..a | X \in b..\max(\mathcal{D})}$$

   ove $a, b \in \mathcal{D}$, e b è l'elemento immediatamente più grande di a in $\mathcal{D}$. Se $\mathcal{D}$ è un intervallo $x..y$ si prenderanno $a = \lfloor (x+y)/2 \rfloor$ e $b = a + 1$.

1. implicazione:

$$\frac{(C_1 \rightarrow C_2)}{\neg C_1 | C_2}$$

2. valore assoluto:

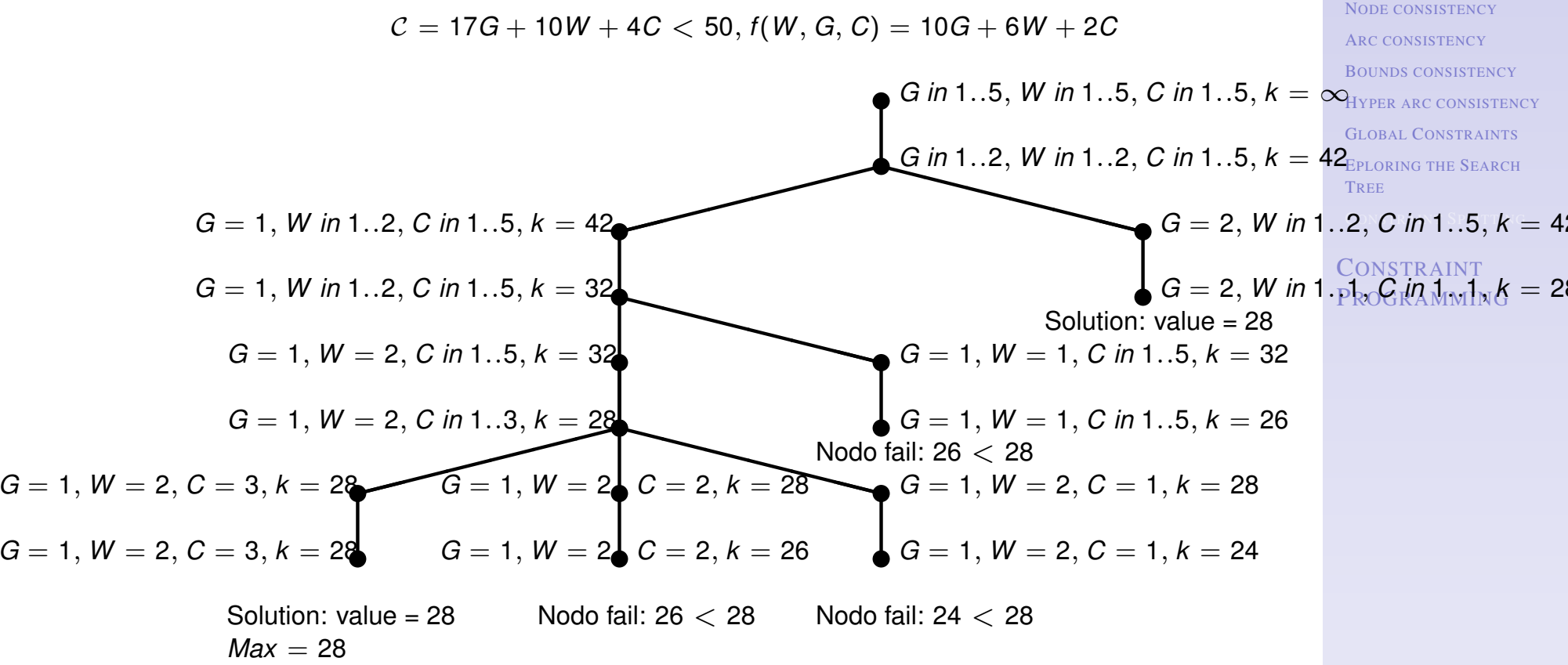
$$\frac{|e| = X}{X = e | X = -e}$$

3. diseguaglianza:

$$\frac{e_1 \neq e_2}{e_1 < e_2 | e_2 < e_1}$$

Constraint Solving

prop-labeling-tree per COP



Constraint Programming

- Si deve disporre di un linguaggio di programmazione con vincoli in diversi domini (interi, Booleani, reali, insiemi, etc)
- Provvisto di un risolutore di vincoli con una serie di parametri per guidare la fase di labeling
- Provvisto anche di costrutti standard dei linguaggi di programmazione
- Allo stato attuale, si trovano (anche liberi)
  - Linguaggi logici (Constraint Logic Programming—CLP)
  - Linguaggi ad-hoc per il modeling (OPL, Zinc, etc)
  - Vincoli in linguaggi piu' tradizionali (p.es., Gecode in C++)
- In queste lezioni ci focalizzeremo sul CLP

Constraint Logic Programming

- Il linguaggio di base è Prolog, provvisto di librerie di vincoli. Useremo la libreria sui domini finiti. Potete usare GNU, Eclipse, B-Prolog (liberi) o SICStus (a pagamento, ma abbiamo le licenze)
- La tecnica di programmazione di base è la constraint & generate.
- La forma generale di un programma è

```
solve(X1,...Xn) :-
      constrain(X1,...,Xn),
      labeling([parametri],[X1,...,Xn]).
```

- `labeling` è built-in. Bisogna definire i vincoli e scegliere dei buoni parametri per la ricerca. Di solito va bene `[ff]`
- In un secondo momento si può ottimizzare il labeling

Constraint Logic Programming
Le N-regine

```
queens(N, Queens) :-
        length(Queens, N),
        domain(Queens,1,N),
        constrain(Queens),
        labeling([], Queens).
constrain(Queens) :-
        alldifferent(Queens),
        diagonal(Queens).
```

CONSTRAINT LOGIC PROGRAMMING

LE N-REGINE

```
diagonal([]).
diagonal([Q|Queens]) :-
        sicure(Q, 1, Queens),
        diagonal(Queens).
sicure(_,_,[]).
sicure(X,D,[Q|Queens]) :-
         nonattacca(X,Q,D),
         D1 is D+1,
         sicure(X,D1,Queens).
nonattacca(X,Y,D) :-
         X + D #\= Y,
         Y + D #\= X.
```